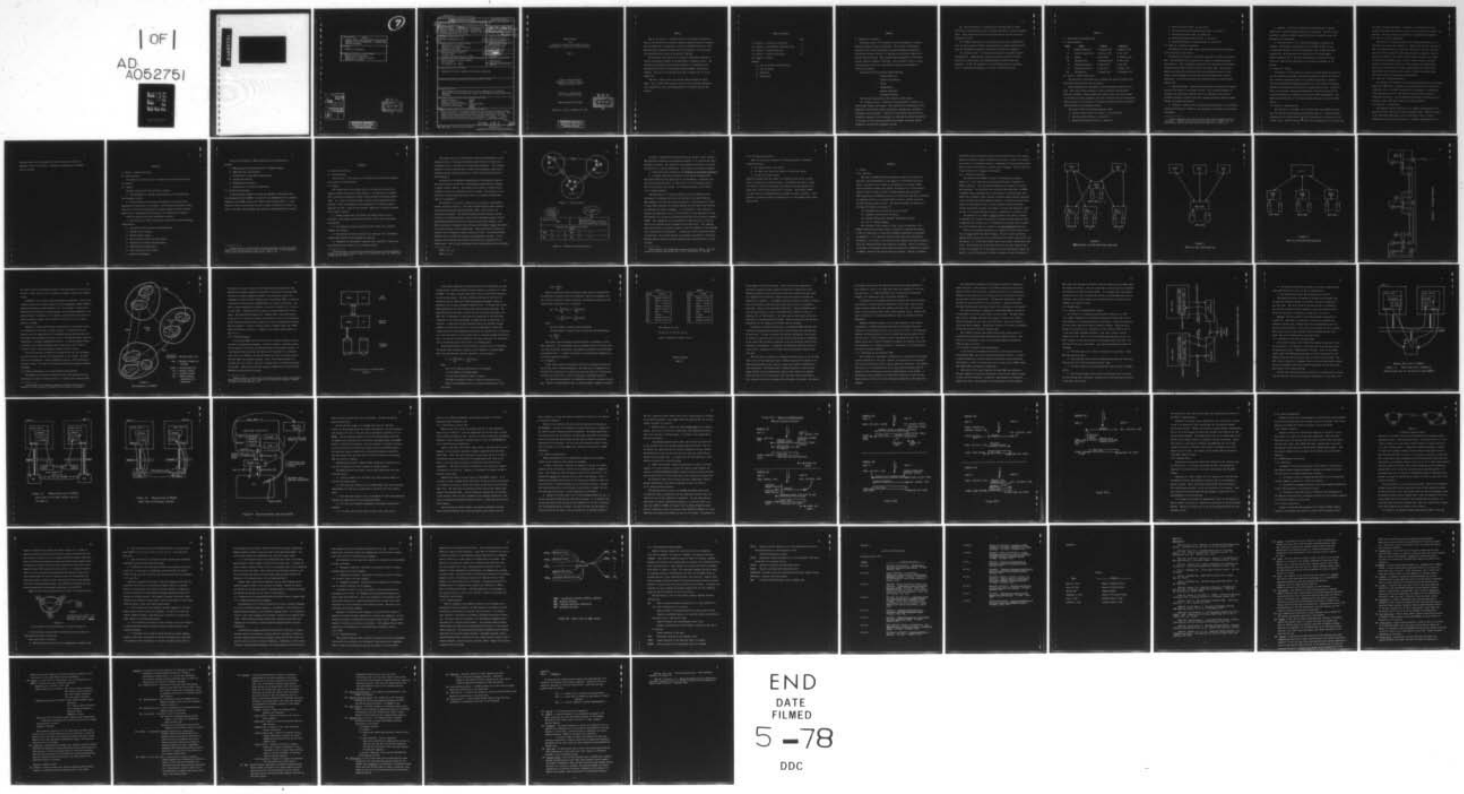
AD-A052 751 KANSAS STATE UNIV MANHATTAN DEPT OF COMPUTER SCIENCE F/G 17/2
PROJECT REPORT FOR FUNCTIONALLY DISTRIBUTED COMPUTER SYSTEMS DE--ETC(U)
OCT 77 V E WALLENTINE DAAG29-76-G-0108
UNCLASSIFIED ARO-13835.1-A-EL-PT-1 NL

| OF |
AD
A052751



END
DATE
FILMED
5-78
DDC

AD A052751

7

4. TITLE (and Subtitle)

PART I

PROJECT REPORT FOR FUNCTIONALLY DISTRIBUTED COMPUTER SYSTEMS DEVELOPMENT: SOFTWARE AND SYSTEMS STRUCTURE

7. AUTHOR(s)

Virgil E. Wallentine

9. PERFORMING ORGANIZATION NAME AND ADDRESS

Kansas State University
Department of Computer Science
Manhattan, KS 66506

NTIS White Section
DDC Buff Section
UNANNOUNCED
JUSTIFICATION
BY
DISTRIBUTION/AVAILABILITY CODES
Dist. AVAIL. and/or SPECIAL
A

DDC
RECEIVED
APR 17 1978
D

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER (18) ARO 13835.1-A-EL-PT-1 (11)	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) PROJECT REPORT FOR FUNCTIONALLY DISTRIBUTED COMPUTER SYSTEMS DEVELOPMENT: SOFTWARE AND SYSTEMS STRUCTURE. Part I. PART I	5. TYPE OF REPORT & PERIOD COVERED (9) Interim rept. 15 Jan 76 - 1 Nov 77 / 15 Jan 76 - 1 Nov 77 / 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) (10) Virgil E. Wallentine	8. CONTRACT OR GRANT NUMBER(s) (15) DAAG 29-76-G-0108 new
9. PERFORMING ORGANIZATION NAME AND ADDRESS Kansas State University Department of Computer Science Manhattan, KS 66506	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS (12) 75 p.
11. CONTROLLING OFFICE NAME AND ADDRESS US Army Research Office P O Box 12211 Research Triangle Park, NC 27700	12. REPORT DATE (12) Oct [redacted] 77 13. NUMBER OF PAGES 72 pages
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) US Army Computer Systems Command Attn: CSCS-AT Ft. Belvoir, VA 22060	15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

The findings in this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computer Network	PASCAL
Computer Message System	DBMS
Mini/Micro Computer Network	Back-End System
Distributed Processing	MIMICS
	KSUBUS

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This is the first of a two-part report of the research performed by Kansas State University in multiple processor computer systems and networks. This report covers the research effort through the design phase. The known problems are defined and alternative solutions are developed. An alternative solution is selected for the building of a prototype network.

391 123

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

PROJECT REPORT

for

Functionally Distributed Computer Systems
Development: Software and Systems Structure

PART I

KANSAS STATE UNIVERSITY
Department of Computer Science
Manhattan, Kansas 66506

Project No. P-13835-A-EL
U.S. Army Research Office

GRANT NO DAAD-29-76-G-0108

January 15, 1976 to September 15, 1977

DDC
RECEIVED
APR 17 1978
D

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

PREFACE

This is the first of a two-part report of the research performed by Kansas State University in multiple processor computer systems and networks. This investigation is supported by a grant of $190,000 from the U.S. Army Research Office, Research Triangle Park, North Carolina. The University has provided matching funds in the amount of $28,383.

The principal investigator is Dr. Virgil E. Wallentine, assisted by faculty and graduate students of the Department of Computer Science. The research was performed at Kansas State University in coordination and cooperation with the U.S. Army Computer Systems Command, Fort Belvoir, Virginia. The term of the research grant was 15 January 1976 to 15 September 1977.

This Part I report covers the research effort through the Design Phase. Part II will follow and will cover the effort through implementation, integration, test, and demonstration of a prototype model of the network.

TABLE OF CONTENTS

CHAPTER 1

1. Overview of the Project

The general nature of the research is the investigation of multiple processor computer systems and networks. The Principal Investigator, assisted by the faculty and graduate research assistants, explored the alternative methods of design of a functionally distributed computer network for data processing. This research takes advantage of the potential of mini- and micro-computer technology. The end product will be a prototype system that serves as a test bed for testing the performance of typical data systems.

The research effort followed a phased approach:

Problem Definition

Solution Alternatives

Design

Implementation

Systems Integration

Prototype Operation

The work was concentrated in four specific problem areas:

(1) Software Utility - Software has been developed to operate in a multi-vendor computer environment. This involved the investigation of the problems of multiple CPU software portability, adaptability, conversion, development, and maintenance. This area of the program concentrated on a comparative analysis of the techniques for achieving the desired portability in the areas of data processing application programs, operating systems dependence, and data base management systems.

(2) Data Accessibility - Techniques have been developed to permit data bases to be distributed across a network accessible to local and regional query. These techniques provide for the protection of the data bases from unauthorized access.

(3) Hardware Specification - Specifications have been established for mini- and micro-computer hardware characteristics to support minimum requirements for performance, security, reliability and fail-soft capability.

(4) Network Control - Research has been done on alternative network configurations. Communications, message processing, and other controls required for system balance and hardware/software interface have been developed. To test the viability of the distributed system, a prototype will be developed and submitted to a series of performance tests.

CHAPTER 2

2. The Technical Development Plan

2.1 Phase Schedule

Phase	Effort	Started	Completed
I	Problem Definition	15 January 1976	1 February 1976
II	Solution Alternatives	2 February 1976	14 May 1976
III	Design	15 May 1976	15 November 1976
IV	Implementation	16 November 1976	14 March 1977
V	Systems Integration	15 March 1977	1 July 1977
VI	Prototype Operation	2 July 1977	1 August 1977
VII	Documentation	15 August 1977	14 September 1977

2.2 Phase I - Problem Definition

The objective of this phase was to identify the specific problem areas upon which research effort must be applied.

Certain problems were identified in the KSU proposal submitted to the Army. There were several alternative areas of research outlined along with those problems. These problems and research areas were then reviewed in the light of recent experience, both here at KSU and in other universities. From the review, new definitions of problems were made and areas of research were described in more definitive terms.

The review covered the following specific areas:

a. Software Utility (Para. II, Section A of the Proposal)

b. Data Accessibility (Para. II, Section B)

c. Hardware Specification (Para. II, Section C)

d. Resources Control (Para. II, Section D)

e. Mini-micro Technology Considerations (Para. V, Section B)

f. Hierarchical Systems (Para. V, Section B)

g. Dynamic Performance Monitor (Para. V, Section C)

h. Distributed Control of Networks (Para. V, Section D)[1]

2.3 Phase II - Solution Alternatives

The objective of this phase was to establish the direction and approach to be used in the design effort.

Consideration was given to the various approaches to achieve the project goal. The approaches were considered in the light of problems identified during Phase I. This consideration resulted in the selection of the research direction believed most likely to achieve the project goal. The selected approach and research direction were to be coordinated with the U.S. Army Computer Systems Command, to insure that follow-on design effort would be appropriate. The solution alternatives provided direction in the following design areas:

a. Network Topology: Alternative configurations of hardware and communications to be tested in a prototype solution. Mini- and microcomputers in networks with large scale computers were to be considered along with the attendent problems of software interface, message processing, network communications and network load balance.

b. Software: Software with and without micro-processors to be considered, along with portability, conversion, and maintenance of software in the network environment.

---

[1]Project Management Plan for Functionally Distributed Computer Systems Development: Software and Systems Structure, Project No. P-13835-A-EL, U.S. Army Research Office, Grant No. DAAD-29-76-G-0108, Feb. 3, 1976, p. 7.

c. Hardware: Technical and functional characteristics of various commercially available computing systems were considered. The use of microprocessors to achieve hardware compatibilities was considered, along with reliability and performance.

d. Data Base System: Use of data base management techniques was proposed. Alternatives included design of a new DBMS suitable for use in distributed networks, use of existing DBMS's such as IDMS (Cullinane Corporation), and hardware/software modification of existing DBMS's. Solution synthesis was to take into consideration the long term desires of the U.S. Army and the constraints of the resources available to the project.

2.4 Phase III - Design

The objective of this phase was to write a detailed design specification for the prototype functionally distributed network and a design specification for a data base management system to run within the prototype. Using the direction and approach developed during Phase II, the Principal Investigator directed the developing and writing of the design specification. During this phase, the Principal Investigator published the "Functional Specification" to provide guidance and continuity to the several Chief Investigators. The DBMS design development and the network design development proceeded independently.

2.5 Phase IV - Implementation

The objective of this phase is to construct and test modules and document the various system designs produced during Phase III. Graduate students working under the supervision of the Chief Investigators will program the prototype system. Modern techniques of structured programming will be followed and

all effort carefully documented. As modules are completed and tested, results and documentation will be reviewed by the Principal Investigator for compliance with basic design and conformity to documentation standards.

2.6 Phase V - Systems Integration

The objective of this phase will be the integrated operation of the prototype test bed and the operation of a DBMS within that test bed environment. As systems modules are completed, they will be tested in operation with other modules using both test data and synthesis programs. When all modules are completed, the system will be tested, corrected, and refined as dictated by performance under a load of various synthetic programs. Synthetic programs will provide the various system load conditions due both to data processing and to system communications and configuration. System modification will be made in order to achieve an optimal system performance under the synthesized load conditions.

2.7 Phase VI - Prototype Operation

The objective of this phase is the successful operation of the prototype system and a DBMS under a synthetic data processing load for collection of statistical data. A synthetic program will be developed as a cooperative effort between KSU and U.S. Army Computer Systems Command, Advanced Technology Directorate. Various host/back-end systems configurations will be tested using a DBMS agreed upon by both KSU and USACSC.

2.8 Phase VII - Documentation

The objective of this phase will be to complete documentation of the prototype system, DBMS, and to collect statistical data. Copies of reports of the individual investigators will be provided to the U.S. Army in accordance with the terms of the grant. A complete specification of the

prototype system will be prepared for delivery as part of the final technical report of the project. Similarly, documentation of the DBMS will be included.

CHAPTER 3

3. Phase I - Problem Definition

3.1 Phase Objective

The objective of this phase was to identify the specific problem areas for research.

3.2 Summary

Two basic problem areas were defined as follows:

a. The development of a design specification for a distributed data base management system.

b. The development of a prototype mini-computer network onto which maxicomputer functions may be off-loaded. In the network development, advantage will be taken of micro-processors to handle network operation overhead. Additionally, consideration will be given to the portability of software.

Areas for specific study were identified as follows:

a. Line protocol for remote system (micro design level) and minicomputer implementation

b. Line protocol (micro-processor implementation)

c. Network control language

d. Network control system

e. Micro interface control for local system

f. Portability of network operating system

g. DBMS intertask communications

h. Local network protocol software

i. Synthetic Programming

A plan for development of DBMS specifications was established as follows:

a. DBMS language specifications based on CODASYL standards

b. DBMS functional specification

c. Incorporation of portability specification

d. Software distribution

e. Communications interface

f. Incorporation of network considerations

3.3 Problems and Solutions

It was originally intended to design and implement a distributed data base management system (DDBMS). In practice, the implementation of a complete, portable, secure DDBMS is beyond the scope of this research effort. It was decided to write specifications for such a system as indicated in Section 3.2 above. But rather than implement that system, an existing DBMS will be used.[1]

---

[1] Progress Report on Functionally Distributed Computer Systems Development: Software and Systems Structure, Project No. P-13835-A-EL, U.S. Army Research Office, Grant No. DAAD-29-76-G-0108, May 20, 1976, p. 45.

CHAPTER 4

4. Solution Alternatives

4.1 Phase Objective

The objective of this phase was to establish the direction and approach to be used in the design effort.

4.2 Summary

The primary goal for the design phase is to define a prototype minicomputer network consisting of three machines. The approach is to proceed with a relatively unrestricted design of both software and hardware mechanisms.[1] As a result of initial studies, several research papers have been produced. These are documented in the appendices of the Progress Report dated May 20, 1976. Based on these studies, approaches to specific problems were formed as follows:

a. Network connectivity requirements and design objectives were described. The hardware and software structure to meet those requirements was proposed.

b. The problems of program portability were studied, and a portable language was proposed.

c. The problems of network usability were examined, and an Extensible-Contractible Network Control Language was proposed.

d. Techniques for performance evaluation were considered. These areas are discussed in more detail in the following section.

---

[1] Progress Report on Functionally Distributed Computer Systems Development: Software and Systems Structure, Project No. P-13835-A-EL, Grant No. DAAD-29-76-G-0108, May 20, 1976, p. 5.

The primary concern in constructing a network of minicomputers is the interconnectivity of machines and software which permits the system to be configured to meet a particular data processing requirement. This configurability can only be achieved when a variety of intermachine connections is available and the associated software to control these network links has been developed.[2]

An approach to intermachine communication is illustrated in Figure 1. The system consists of clusters of minicomputers interconnected by memory-to-memory adapters (MMA's). These MMA's, which operate at memory speeds, eliminate the need for packetizing within a cluster. The clusters are connected via a packet-switching system, since it is assumed a minimal data capacity is necessary.[3]

The approach to the control software can be viewed as a hierarchical system, as shown in Figure 2. The design includes several vendor supplied local operating systems (Extended Machines (EMs)) which control the heterogeneous machines. The interface between network resources and EMs resources is the Network Resource Control which transforms network resource allocation/deallocation into local operating system (EMs) commands. The NRC is connected to requestors (User Processes and Command Language Processors) of network resources via the network IPCS. These requests will achieve the process, file, and general resource allocation control. The Network Machine handles batch or interactive requests and synchronizes job steps. In the produced, only data base and program construction user processes will be demonstrated. This is, only rudimentary NM and NRC functions will be developed.

---

[2] Ibid., pp. 2, 3.

[3] Ibid., p. 15.

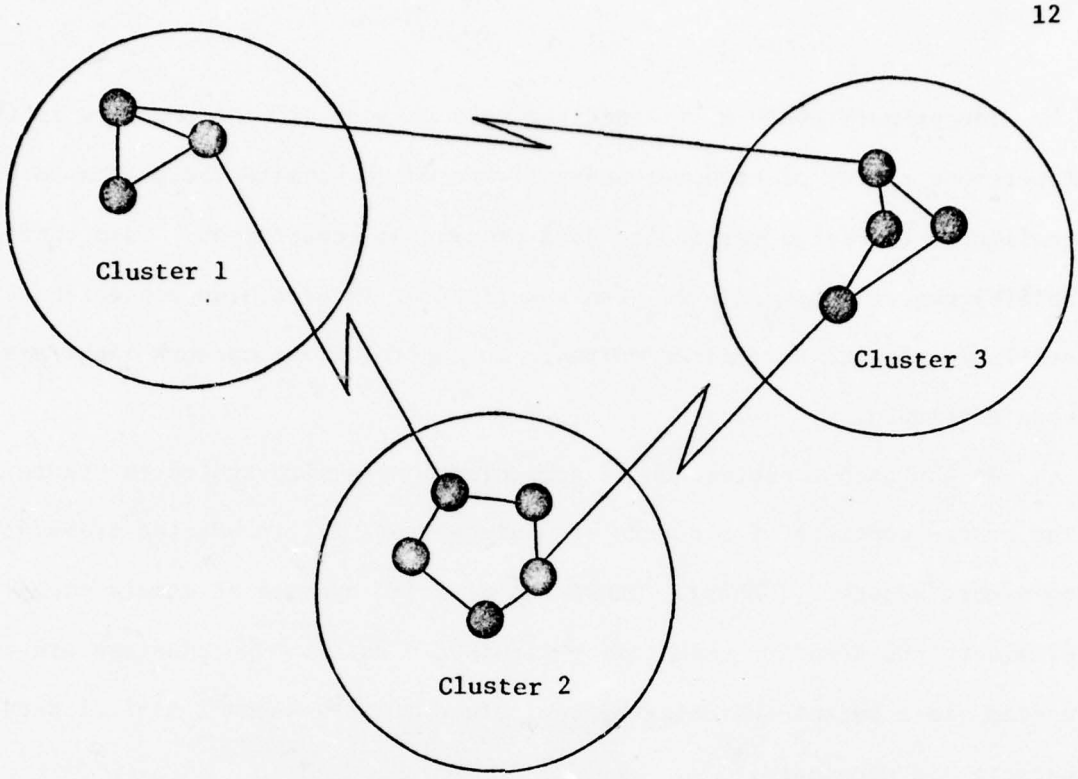


Figure 1. Cluster Network

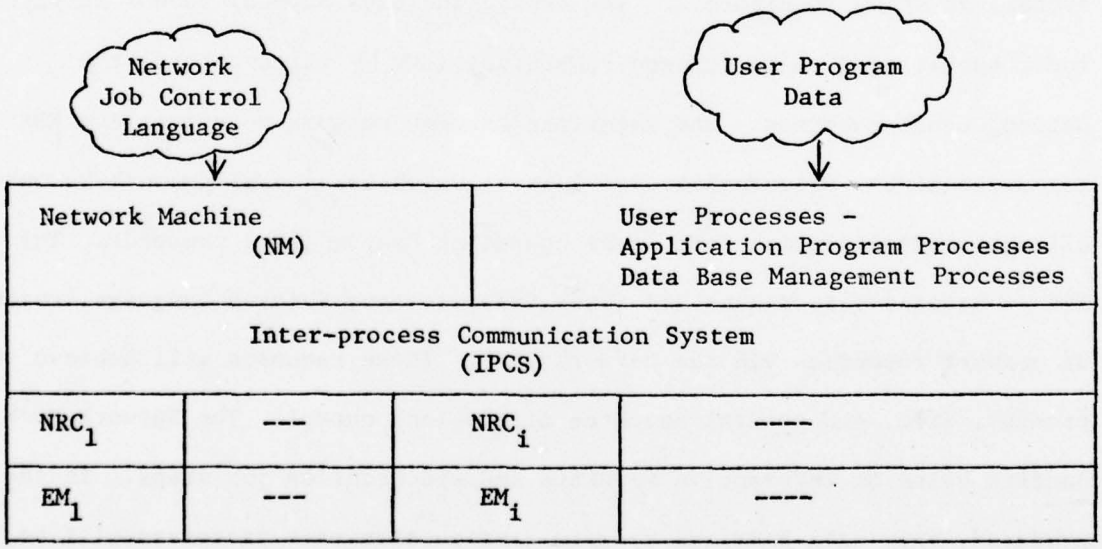


Figure 2. Hierarchy of Network Software

In order to functionally distribute data base systems, control software, and application programs to heterogeneous machines, it is critical that these programs be portable. The approach to the portability problem is to include portability as an initial design goal. The design for portability includes:

a. Structured design (reported in the Workbook on Structured Programming)

b. A portable high-level language is to be used in implementation.

Concurrent PASCAL has been chosen due to its portability, concurrency, and job control characteristics.[4] We will "port" it to the Interdata 8/32 to test its portability and utility. It is already available on the PDP-11.

4.3.3 Network Usability

The utilization of a computer network is not only dependent on the utilization of programs, but also on the utility of its user interface. When computers with different operating systems or architectures are linked to form complex networks for resource sharing, the user must become familiar with multiple job control languages. It is desirable to link computers to provide new capabilities, but it is not desirable to add complexity to usage. The approach is to design an Extensible-Contractible Network Control Language (ENCL). This language is to be extensible to permit the user to create an easy-to-use, powerful control language for his application. This language must also provide a contraction feature so that the translator of the language can be constructed to be efficient. A second use of the contraction feature is protection. The systems security can alleviate the ability to access and possibly destroy certain protected resources from a particular user or class of users.

---

[4] Brinch Hansen, The Programming Language--Concurrent PASCAL. IEE Transactions on Software Engineering, Vol. 1, No 2 (June 1975), pp. 199-207.

4.3.4 Performance Evaluation

There are three basic reasons for conducting computer performance evaluation studies:

a. To size and select a new system

b. To predict the effects of changes in the present system

c. To tune the present system

Synthetic programs have been shown to be valuable tools in all of these types of performance evaluation studies. The approach has been to survey the current literature and examine the elements of current synthetic programs which could be distributed across a network. Then several models were developed for extending synthetic programs to various networks. This approach is further discussed in Appendix 6.2.2 of the Progress Report dated May 20, 1976.

CHAPTER 5

5. Design

5.0.1 Overview

The basis for MIMICS (Mini-Micro-Computer System) is the utility of both mini- and microcomputers in the support of a distributed data base system. The goal of the research is development of a prototype MIMICS which accommodates hardware and software interconnection of heterogeneous computers under a common (portable) command language (and structure). This language must accommodate the user at his own level and permit allocation of computing resources in a unified manner using the software connection (network message system) facility. This report documents our approach to the design of MIMICS in the areas of--

(1) mechanisms for accessing data in the network;

(2) hardware interconnection facilities;

(3) network inter-process (message) communication system;

(4) network command language.

The structure of this chapter is first to give an overview of the MIMICS architecture, then the concepts, followed by a detailed description. A table of definitions of terms used in MIMICS documentation is presented at Appendix C. It is important that the reader study this table before proceeding. In section 5.1 the results of our research into design considerations in a distributed data base system are presented. Section 5.2 contains an overview of the message system (network inter-process communication system) in MIMICS. Both user and system views are presented. Section 5.3 presents

the MIMICS hardware architecture which has been developed for large capacity computer-to-computer (memory-to-memory) data transfer. Section 5.4 contains an overview of a preliminary design of a distributed network operating system under a workload described in a network control language. Both the structure of the design and the language are presented.

5.0.2 Introductory Concepts

The motivation for MIMICS is support of distributed processing. Figures 3, 4, and 5 show several topologies which must be supported in MIMICS technology. The links between computers are tradeoffs of economy and performance. The objectives are to provide high-speed buses (a KSUBUS in MIMICS) between geographically close machines, to provide direct slow-speed links (telephone lines--switched or leased) between machines geographically separated but which communicate heavily, to provide slow-speed links (which are used lightly) which are very economical (store-and-forward through intermediate machines), and finally to permit off-loading of a good portion of the network message system on a communications controller (CC).

The approach is to support a cluster topology (see Figure 1) wherein all machines in a cluster are connected by memory-to-memory adapters across a high-speed bus (transfer rate is 10 Megabytes per second), then to permit one or more machines (CC) in a cluster to be store-and-forward functions.

A view of the internal structure of a cluster is shown in Figure 6 with four (4) hosts (H1-H4) and three (3) CC's. The CC's contain the network message system and may serve one or more hosts via a direct connection to the host memory. It is this link (double lines in the figure) through which data flows. The hosts and the CC and the various CC's are also connected via a control line (single line in the figure) over which coordination signals are passed. Data can flow directly between the memories of any two machines in

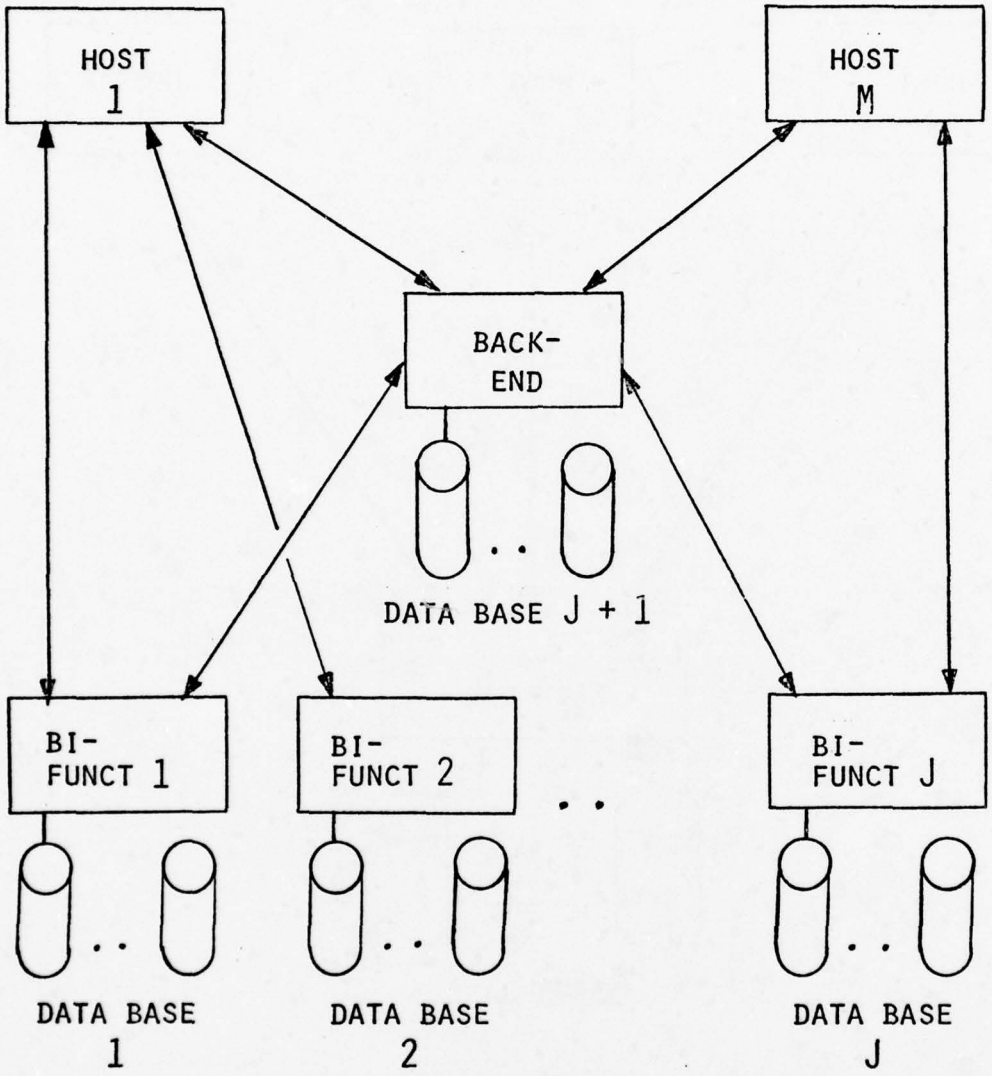


FIGURE 3

DBMS NETWORK WITH BI-FUNCTIONAL MACHINES

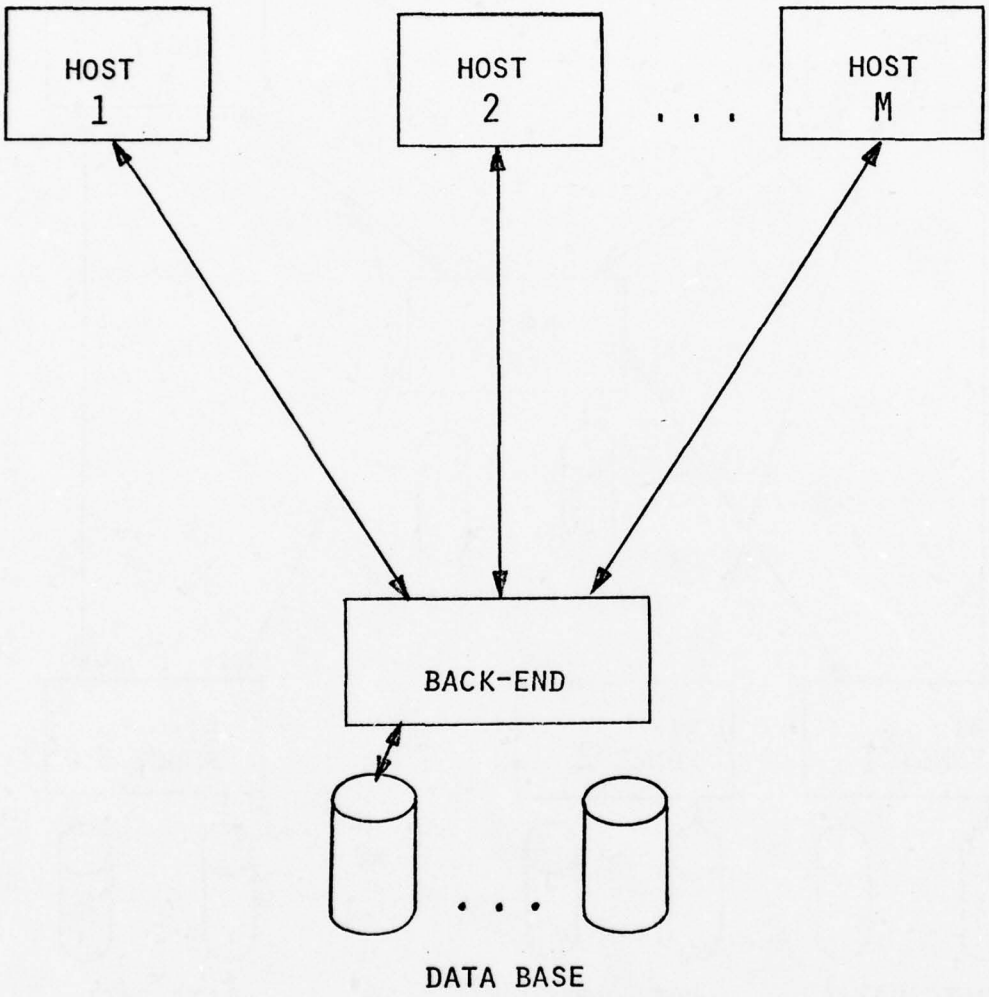


FIGURE 4

MULTIPLE HOST CONFIGURATION

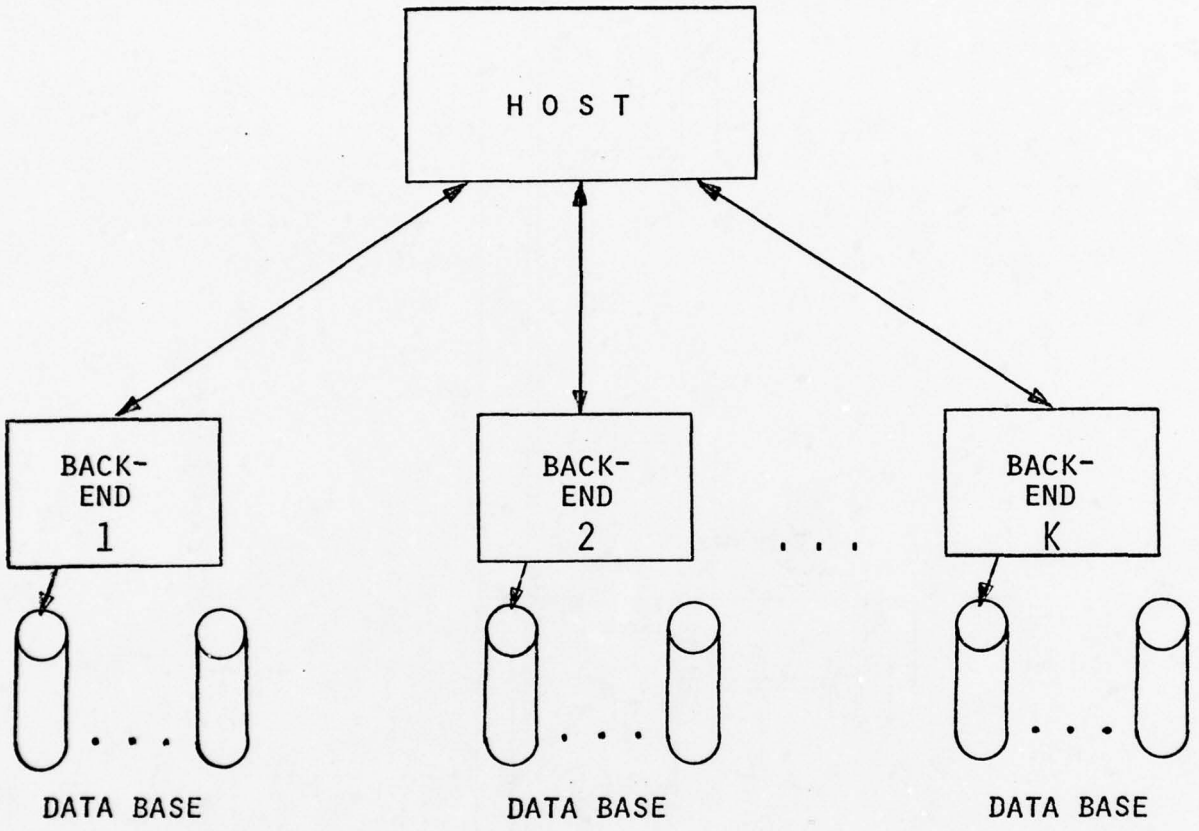


FIGURE 5

MULTIPLE BACK-END CONFIGURATION

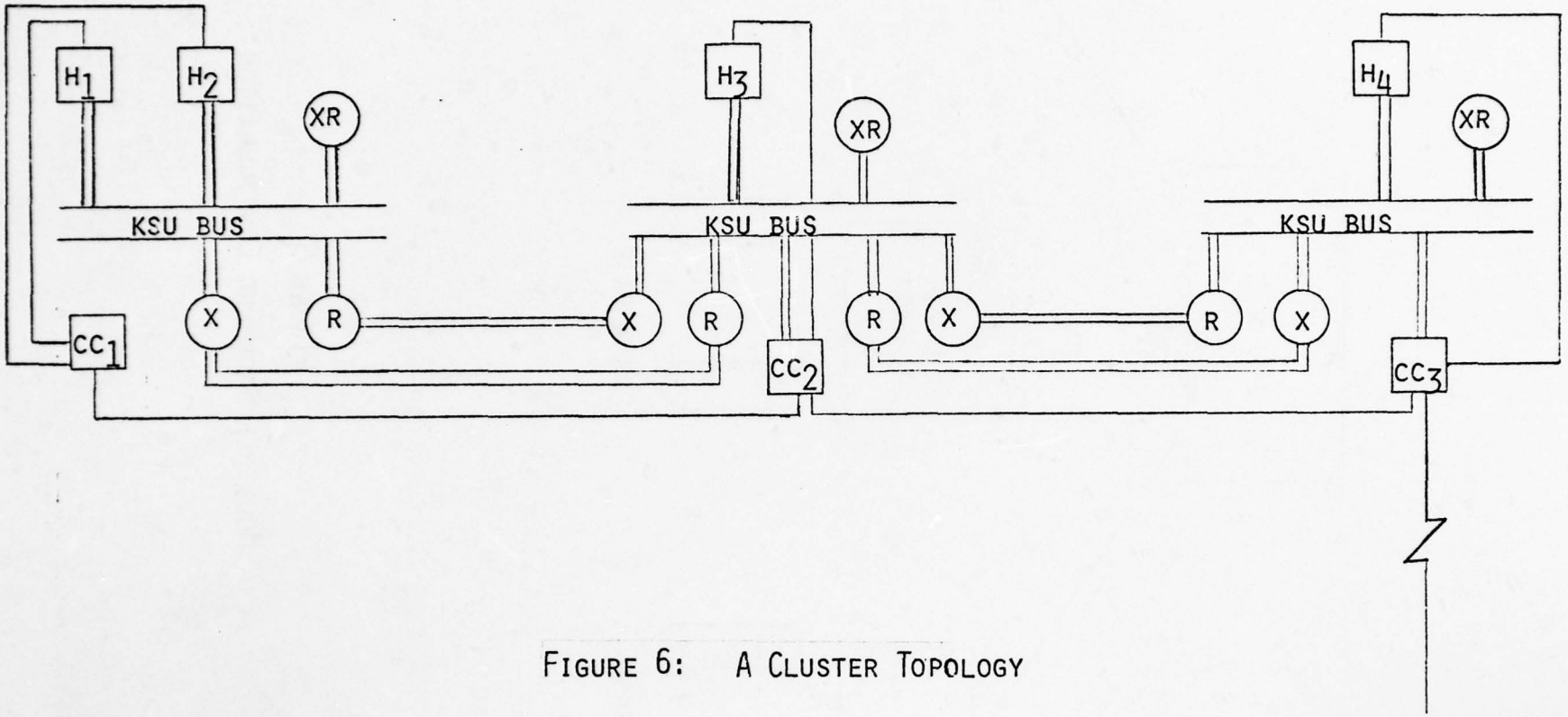


FIGURE 6: A CLUSTER TOPOLOGY

the cluster under the coordinated control of the data movers (X's and R's) by the CC's. Entry and exit from the cluster is through a remote line attached to CC_3.

In MIMICS a full resource sharing philosophy is supported. That is, all resources such as file system, editors, data base management systems (DBMS's), application programs, systems programs, etc. are processes. Resources are requested (and utilized) by sending messages (and receiving responses) to the process which is the resource. For example, an application stores data in a file by sending a message, which consists of that file, to a file system process or a DBMS.

Support for network inter-process communication via the message system is a hierarchy of functions, as shown in Figure 7. Processes are to be executed under the supervision of a Distributed Network Operating System.** The message system consists of MS, PS, and CS. MS (message system) supervises coordination of message "handshaking" protocol. It uses PS (packet system) to packetize and transmit messages across remote lines, and it uses CS (cluster system) to move data memory-to-memory within a cluster.

The first prototype of MIMICS will include the MS, PS, and CS elements executing on three clusters with three machines in one cluster. The DNOS will not be implemented. However, MIMICS will support a fully distributed data base system. The rest of this document details the individual elements of MIMICS.

5.1 Design Considerations for a Distributed Data Base System

The emphasis of the data base portion of the G-108 research has been placed upon the study of design problems. In previous work conducted under

---

**The design of the elements of DNOS are described in Section 5.4. However, the first prototype of MIMICS will not contain this element.

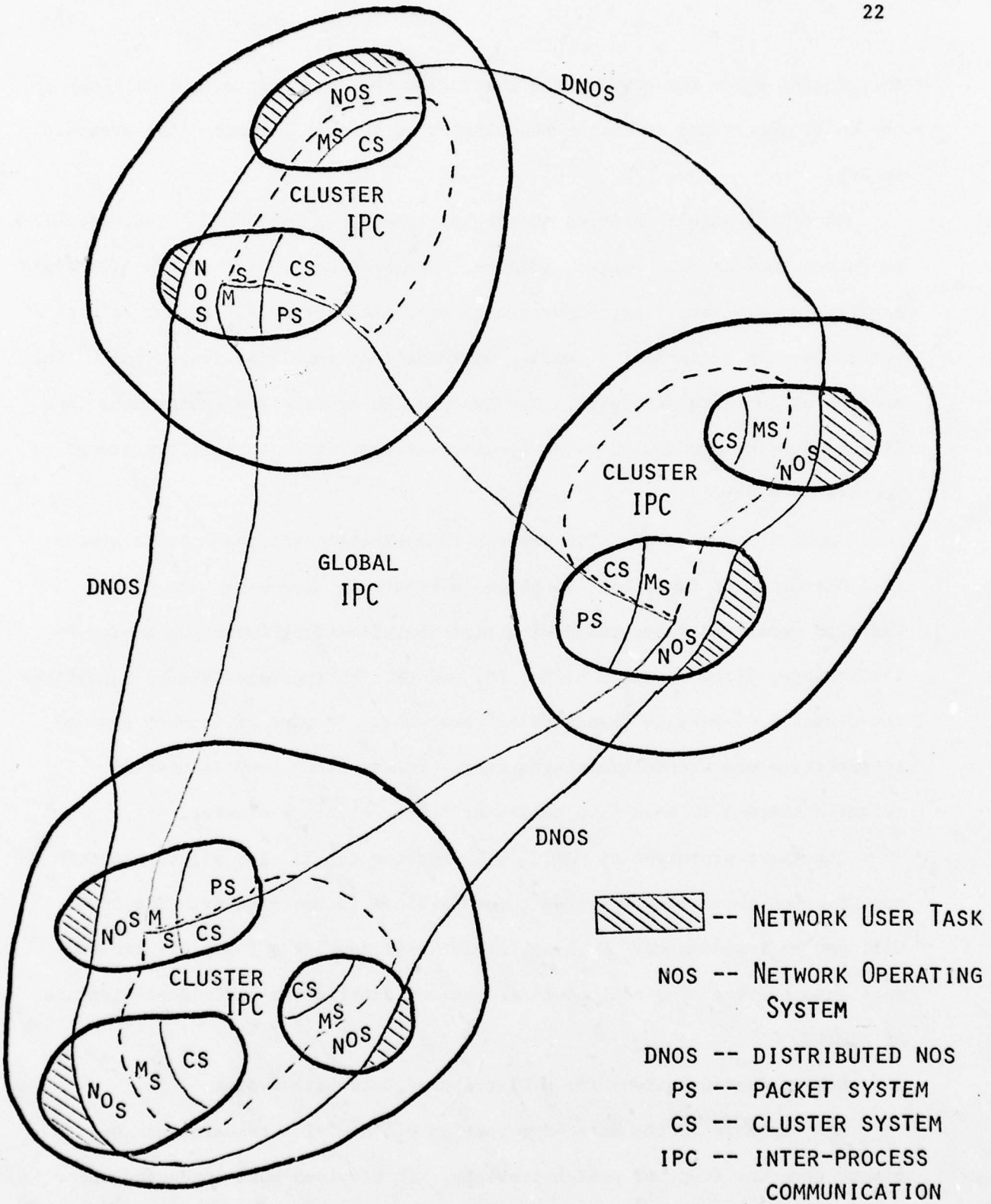


FIGURE 7

DISTRIBUTION OF MIMICS

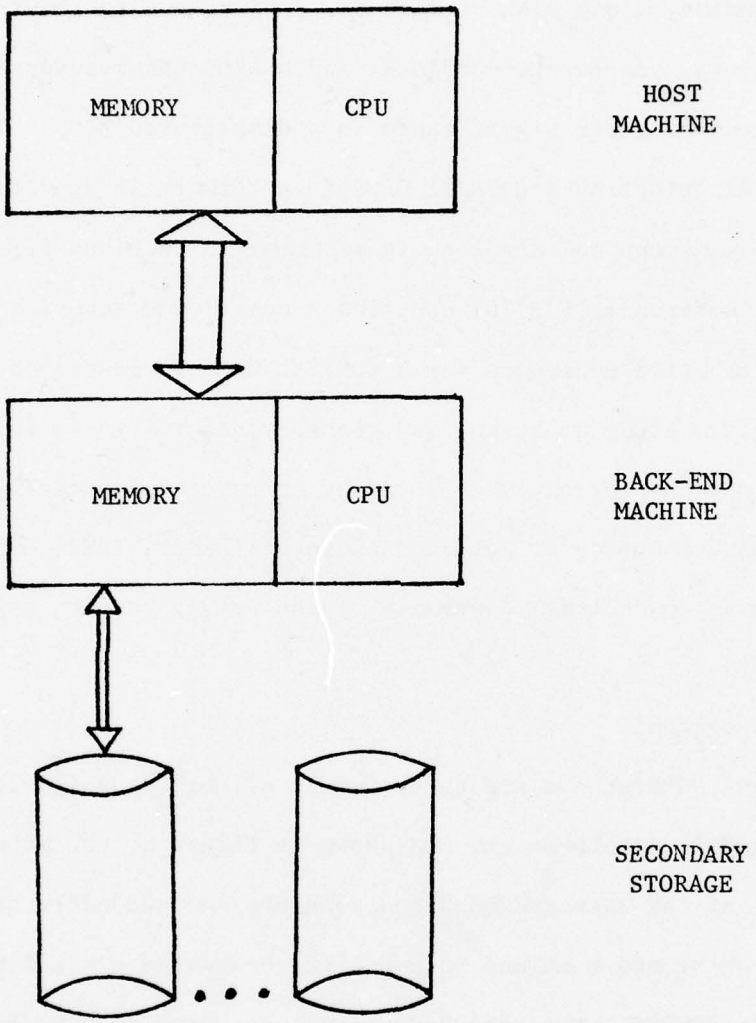
this grant, the basic structure of the back-end and distributed data base systems was presented, along with performance projections for back-end DBMS. The topics of memory management, deadlock, and backup and recovery are considered in terms of their significance in a distributed DBMS. In addition, a mechanism for distributing a CODASYL DBMS in a network is specified. The work on memory management and deadlock is outlined in Sections 5.1.1 and 5.1.2 of this report. Reference [FJM 76] contains a detailed discussion of this topic. The distribution mechanism for a CODASYL DBMS is described briefly in Section 5.1.3. The study of backup and recovery has led to an investigation of the recovery methods currently employed by users of commercially available data base systems. A survey of selected users of ADABAS, IDMS, IMS, SYSTEM 2000, and TOTAL was conducted. A summary of the survey results appears in Section 5.1.4.

5.1.1 Memory Management

In a back-end DBMS, there are three levels of memory available for data accessed by an application program. As shown in Figure 8, the memory levels are the memories of the host and back-end machines and secondary storage. In a configuration where the host and back-end are connected via a high-speed link, such as the KSUBUS[1], the back-end memory can serve as a buffer between the host memory and secondary storage. Pages in the back-end memory are retained until the buffer is full and a new page must be obtained from secondary storage. Pages are written onto secondary storage only if they have been modified by the application program.

---

[1]Progress Report on Functionally Distributed Computer Systems Development: Software and Systems Structure, Project No. P-13835-A-E-L, Grant no. DAAD-29-76-G-0108, Dated December 30, 1976.



Memory Hierarchy in a Back-End DBMS

Figure 8

In the report appearing in Reference [FJM 76], the performance in terms of page faults involving secondary storage transfers is analyzed for a back-end DBMS. The report considers page faults both with and without buffering in the back-end memory. The exact difference between the two-level (no buffering) and the three-level (buffering) memory management schemes is dependent upon the page replacement algorithm used. An optimal page replacement algorithm for the three-level memory system has been developed. The algorithm replaces the page in memory that has the lowest replacement cost. This cost of replacement is based upon the probability of the page being referenced in the future and whether the page has been modified since its retrieval. A page that has been modified requires two secondary storage transfers (a write and a read) if it is to be retrieved at a later time. A page that has not been modified requires only one transfer if it is accessed at a later time since it is not written back to secondary storage upon replacement. For the case of equal probability of future reference, the replacement cost of a modified page is twice that of an unchanged page.

If the probabilities of future reference are known or can be estimated, the optimal page replacement algorithm can be applied in a back-end DBMS. Under this algorithm the expected replacement cost per page is--

$$C = \left(B - \sum_{i=m}^{n} p^2(i)/B\right)\left(1 + \sum_{i=m}^{n} p(i)p(W_i)\right)$$

where

p(i) is the reference probability of the ith page;

n is the number of available pages;

m is the size of back-end memory in pages;

the pages are ranked in order of expected replacement cost;

$p(W_i)$ is the probability of the ith page being rewritten to the data base;

$$\text{and } B = \sum_{i=m}^{n} p(i).$$

When the three-level memory management scheme using the optimal back-end algorithm is compared with the conventional two-level arrangement with Aho, Denning, and Ullman's optimal algorithm, the following reduction in the expected page replacement cost is obtained:

$$C_R = \left(B_h - \sum_{i=m_h}^{n} p^2(i)/B_h\right)\left(1 + \sum_{i=m_h}^{n} p(i)p(W_i)\right)$$

$$- \left(B - \sum_{i=m}^{n} p^2(i)/B\right)\left(1 + \sum_{i=m}^{n} p(i)p(W_i)\right)$$

where

m_h is the number of pages in the host memory;

m is the number of pages in both the host and back-end memories;

$$B_h = \sum_{i=m_h}^{n} p(i).$$

The optimal page replacement algorithm requires a knowledge of future page references. For a reasonably static environment such as the daily cycle of a data processing installation, page behavior information can be obtained from journal files. A complete discussion of back-end memory management can be found in Reference [FJM 76].

5.1.2 Deadlock

In any system involving shared data access, the possibility exists for two or more tasks to become deadlocked. Two tasks are in a deadlock state if each task is waiting for a resource (in this case some unit of data) that is held by the other task. Figure 9 illustrates a deadlock situation.

In a data base environment, the likelihood of deadlock occurring is quite low. However, a mechanism must exist to properly handle a deadlock situation

Task K

Step	Action
K_0	Request record A
K_1	Receive record A
K_2	Lock record A
K_3	Request record B
K_4	Receive record B
K_5	Lock record B
K_6	Process
K_7	Release record A
K_8	Release record B

Task M

Step	Action
M_0	Request record B
M_1	Receive record B
M_2	Lock record B
M_3	Request record A
M_4	Receive record A
M_5	Lock record A
M_6	Process
M_7	Release record B
M_8	Release record A

The sequence of steps

$K_0, M_0, K_1, K_2, M_1, M_2, K_3, M_3$

leads to deadlock of Tasks K and M.

Deadlock Example

Figure 9

if the DBMS is to function properly. There are two basic approaches to solving the deadlock problem--deadlock prevention or deadlock detection. Deadlock prevention involves delaying any data base request that may result in a deadlock until sufficient resources have been freed to preclude the possibility of deadlock. In a deadlock detection scheme, no action is required until the deadlocked tasks have been denied data access for an unusually long time. (That is, the tasks have "timed out.") Once the deadlock condition has been detected, one of the blocked tasks is forced to yield its resources (it is "rolled back"). The other task is then free to access the data which it had previously been denied. When this task frees the resources responsible for the deadlock, the blocked task is restarted.

An analysis of deadlock prevention in a distributed DBMS indicates that information regarding the potential data access conflicts of each task must be maintained. In order to maintain the conflict information, the creation or demise of a data base task at any node in the network must be transmitted to all other tasks in order to determine if any new potential conflicts have developed or disappeared. In addition, the locking or unlocking of data by a task must be transmitted to all tasks that have potential conflicts with this task.

The only type of overhead for a deadlock detection policy is the "rolling back" of one of the deadlocked tasks. Rollback involves the reversal of all operations performed by that task up to some point (either a checkpoint or task initiation). The problem with a rollback operation is that the task being reversed may have modified data used by other tasks in the network. Since this data is no longer valid, these tasks must also be rolled back. This procedure can have a cascading effect throughout the system. The process

of rollback requires that the portion of the data base being modified by the rollback be closed to all other users until the completion of the rollback operation. In a DBMS distributed over a large area with low-speed linkages, the communication delay can become prohibitive.

A comparison of the overhead involved in deadlock detection and prevention produces a fairly constant overhead and in many cases results in preventive measures being taken when deadlock would not occur. On the other hand, a detection scheme has no effect unless deadlock occurs. However, the recovery procedure in a distributed DBMS can result in substantial system degradation.

Deadlock prevention appears to be the safer approach to the problem. Consequently, a deadlock prevention algorithm for distributed data base systems has been developed as part of our data base research. The algorithm involves the identification of a set of records that may be updated by more than one task. A list of shared records is maintained for each task. The shared record list is used to determine if requests for locking a record may lead immediately to a deadlock situation. The details of the algorithm may be found in Reference [FJM 76].

5.1.3 Mechanisms for Distributing a DBMS

The technique for operating on a data base in a host-back-end environment is well established at this time. However, the extension to an environment with several back-end processors requires additional facilities. The problems that arise are the identifications of the proper back-end machine and the movement of data and processor functions among machines in the network. A mechanism has been developed for allowing distributed data access with the exact physical location of the data remaining transparent to the application program.

The distribution mechanism is based upon the concept of logical processor functions. Each processor in the DBMS network may act as host or back-end machine with the ability to execute a certain set of application programs (a host function) or to control access to a particular portion of the data base (a back-end function). The Data Base Administrator (DBA) assigns the logical functions to the processors. A Logical-to-Physical Processor Map maintains the current status of the processors in the network.

The actual execution of commands in a distributed DBMS is controlled by information in the ALL (Area Logical Location) Table. The DMCL (Device Media Control Language) indicates the location in terms of its logical back-end processor for each area in the data base. The ALL is created by processing DMCL statements. The physical location of the data is determined through the Logical-to-Physical Processor Map.

The problems of moving data among machines under system control or by physically transferring disk packs have been studied carefully. Procedures for the movement of data in a distributed DBMS are provided in Technical Report CS 76-22.

5.1.4 Survey of Commercial Data Base Systems

In the interest of studying the problems of backup and recovery in a distributed DBMS, the need for practical experience was felt. In order to gain information on recovery methods as practices in data processing environments, a survey of 15 data base users (3 users each of ADABAS, ADMS, IMS, SYSTEM 2000, and TOTAL) was undertaken.

The results of the survey indicate that most DBMS users attempt to maintain simplicity of both the operational as well as the software aspects of their recovery operations. In general, they minimize the interaction between data bases, thus reducing the cascading effect of task rollback.

Most users rely upon special software (often not supplied by the DBMS vendor) for backup and recovery. They are justifiably concerned with the reliability and integrity of their data base systems. It is apparent that considerable research into the topic of backup and recovery for distributed data bases is required. Work on this subject has been initiated and the results will be compiled in a technical report.

5.2 Message System

5.2.1 Purpose of the MIMICS Message System

The purpose of the message system (hereafter referred to as "MS") is to support communication of messages between user tasks, either in the same machine or in connected machines in the network. Figure 10 illustrates two user tasks, arbitrarily shown in distinct machines. Each task has a message area consisting of a command data buffer (maximum 128K bytes) and a text data buffer (maximum 64K bytes). User task 1 issues a receive request to its MS. Then, if all the necessary preconditions are met, the MS's cooperate to copy the contents of the message buffers from user 1 into the areas in the user 2 data space. The various preconditions include the following:

a. Both tasks have to be correctly "connected" to each other. (Both MS's must agree on this.)

b. Both tasks must have correctly identified both their own "task·port" name and the other "cluster·machine.task·port" name.

c. The send_request and receive_request must agree in mode of communication.

d. The receive message buffers must be of sufficient size to receive the sent message parts (although a message need not have both parts--one part or the other can be null).

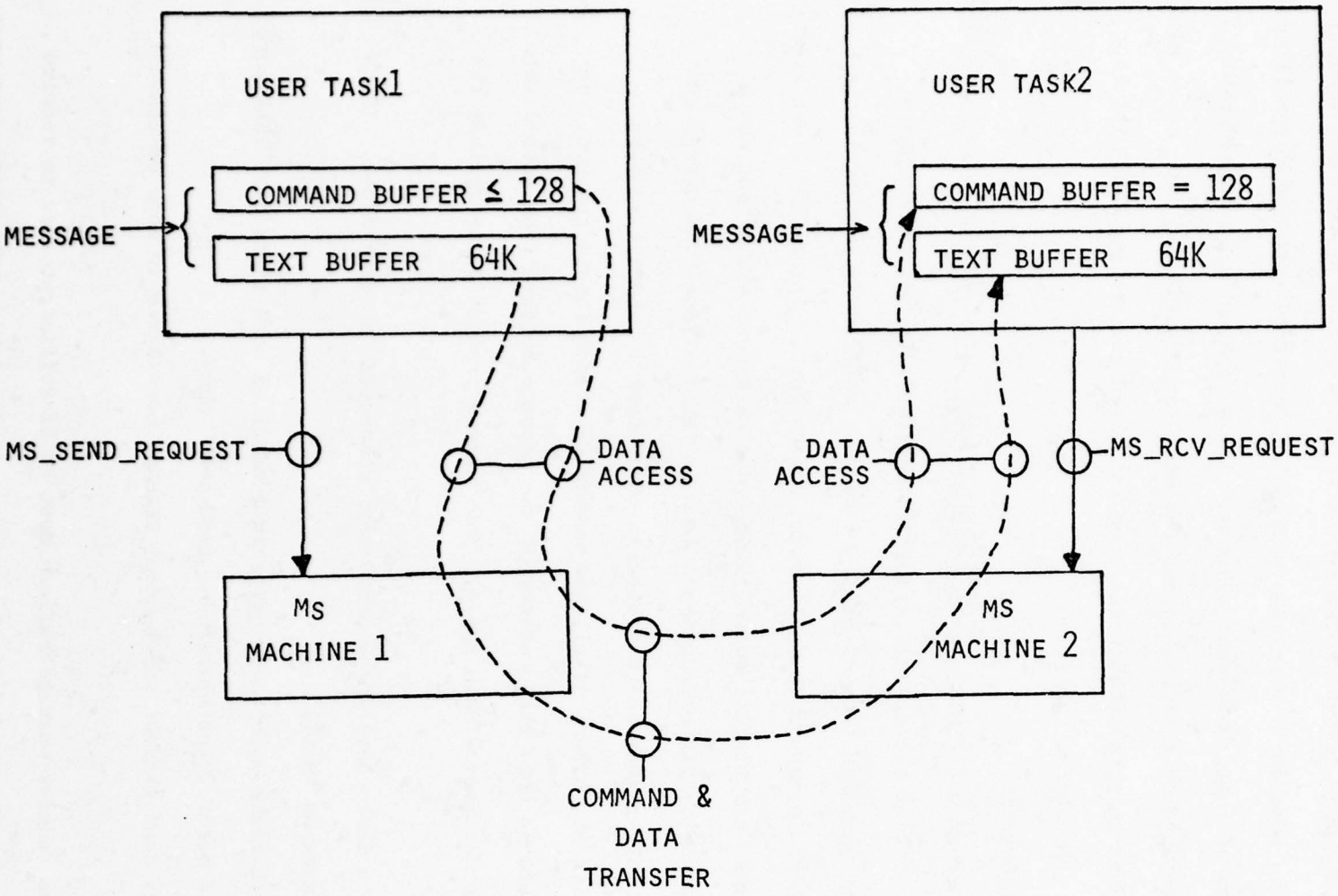


FIGURE 10: COMMUNICATION BETWEEN USER TASKS

e. The requests must have been issued in the proper synchronization time frame (which varies with the mode of communication).

f. The network path between MS1 and MS2 must be functional.

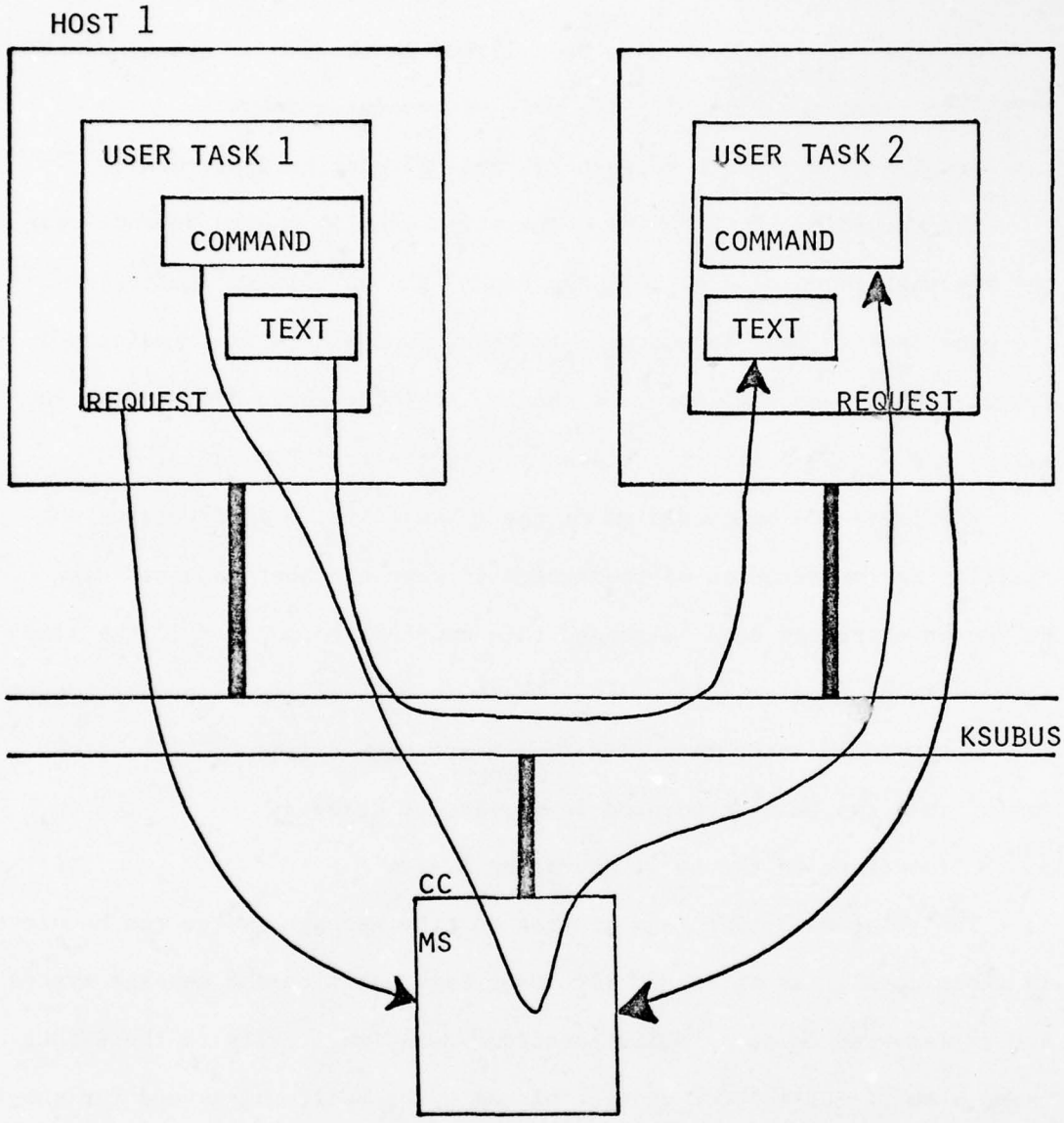
The mechanism by which the message is actually copied depends upon the physical connection between the two MS's, but this is immaterial to the user tasks. They do not need to know how the data is transferred. For the sake of general interest though, at least three different mechanisms are involved, as illustrated in Figures 11, 12, and 13.

Warning: The basic design of the message system incorporates no facility for translation of characters in case the host machines use different character sets, although this can be incorporated in the line drivers in particular MS's. Also, there is no facility to pad or compress data bytes in the case that different hosts have different size bytes. Again, that can be incorporated in particular cases.

5.2.2 Interface to the Local Operating System

The relation between a user task and the message system can be viewed in finer detail, as in Figure 14. User task calls to the message system are implemented using a "SYSQUE_monitor" function. Calls to the SYSQUE cause a small SCB (SYSQUE_control_block) to be built and queued for subsequent receipt by some other task, such as the MS or the user task. The SCB contains a pointer to a parameter block in the user space which contains the actual parameter values for the MS call and space for the result parameter values to be stored by the MS.

Certain other system-type support functions are required (such as initiation of tasks and clean up after abnormal termination of a user task). For



MESSAGE DATA FLOW IN MIMICS:

FIGURE 11: USER TASKS WITH A COMMON CC.

(EITHER SAME HOST OR TWO HOSTS ON SAME KSUBUS)

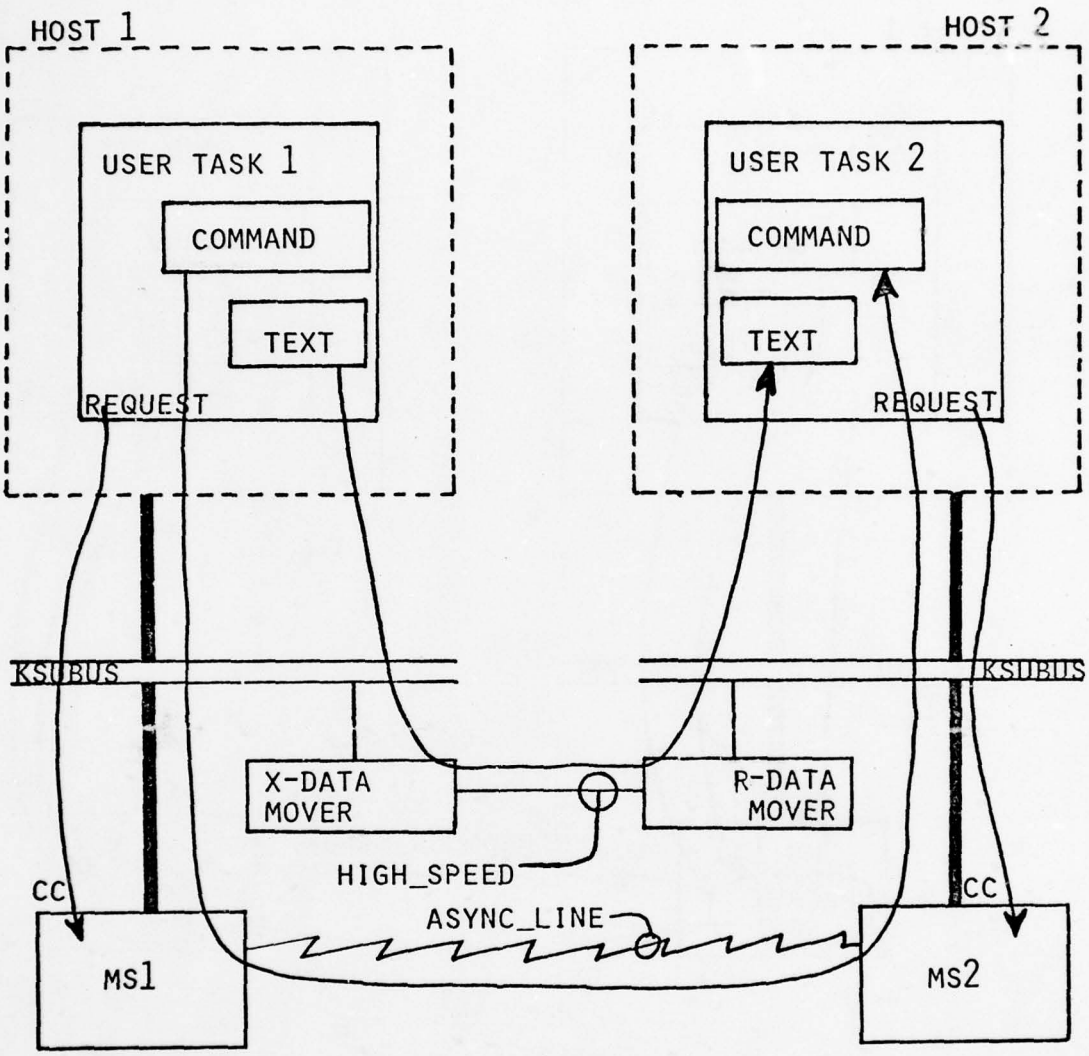


FIGURE 12: MESSAGE DATA FLOW IN MIMICS: USER TASKS IN THE SAME CLUSTER, BUT NOT THE SAME CC.

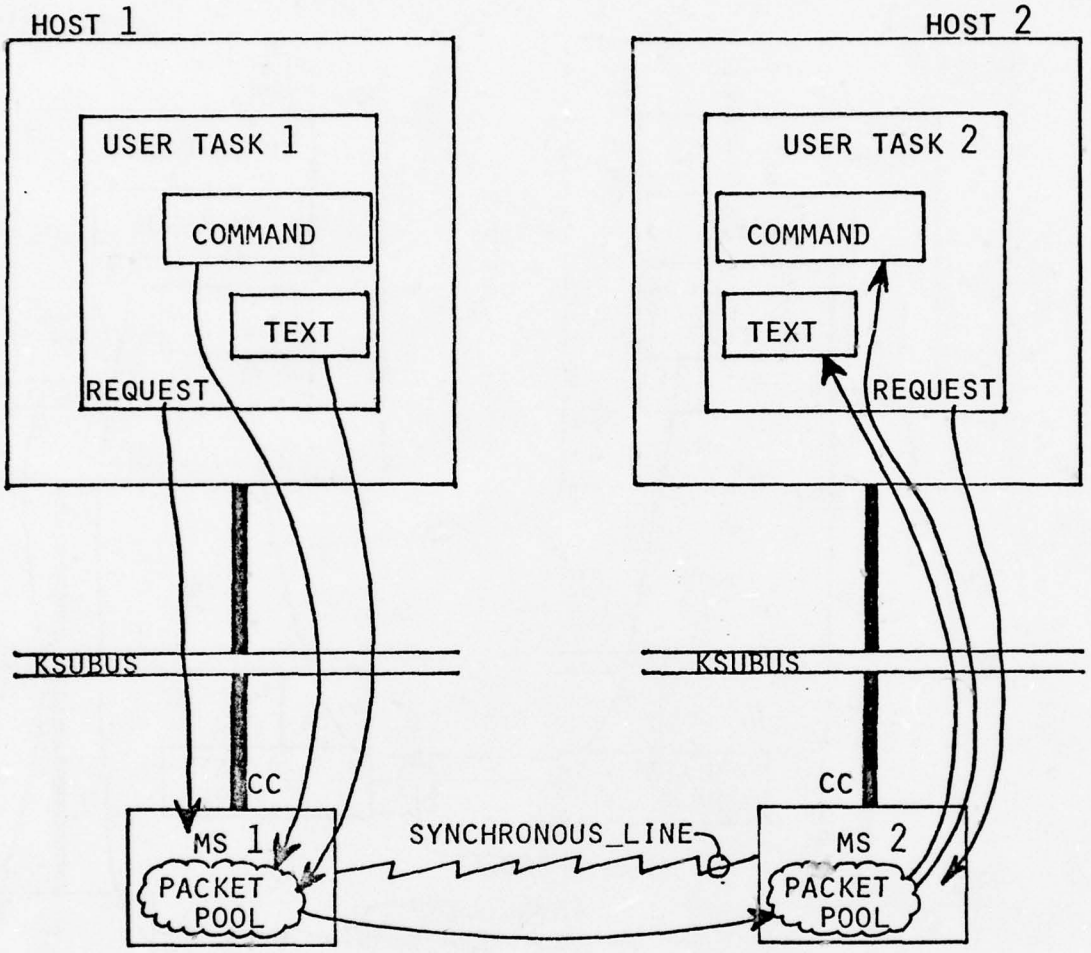


FIGURE 13: MESSAGE FLOW IN MIMICS:
USER TASKS IN DIFFERENT CLUSTERS

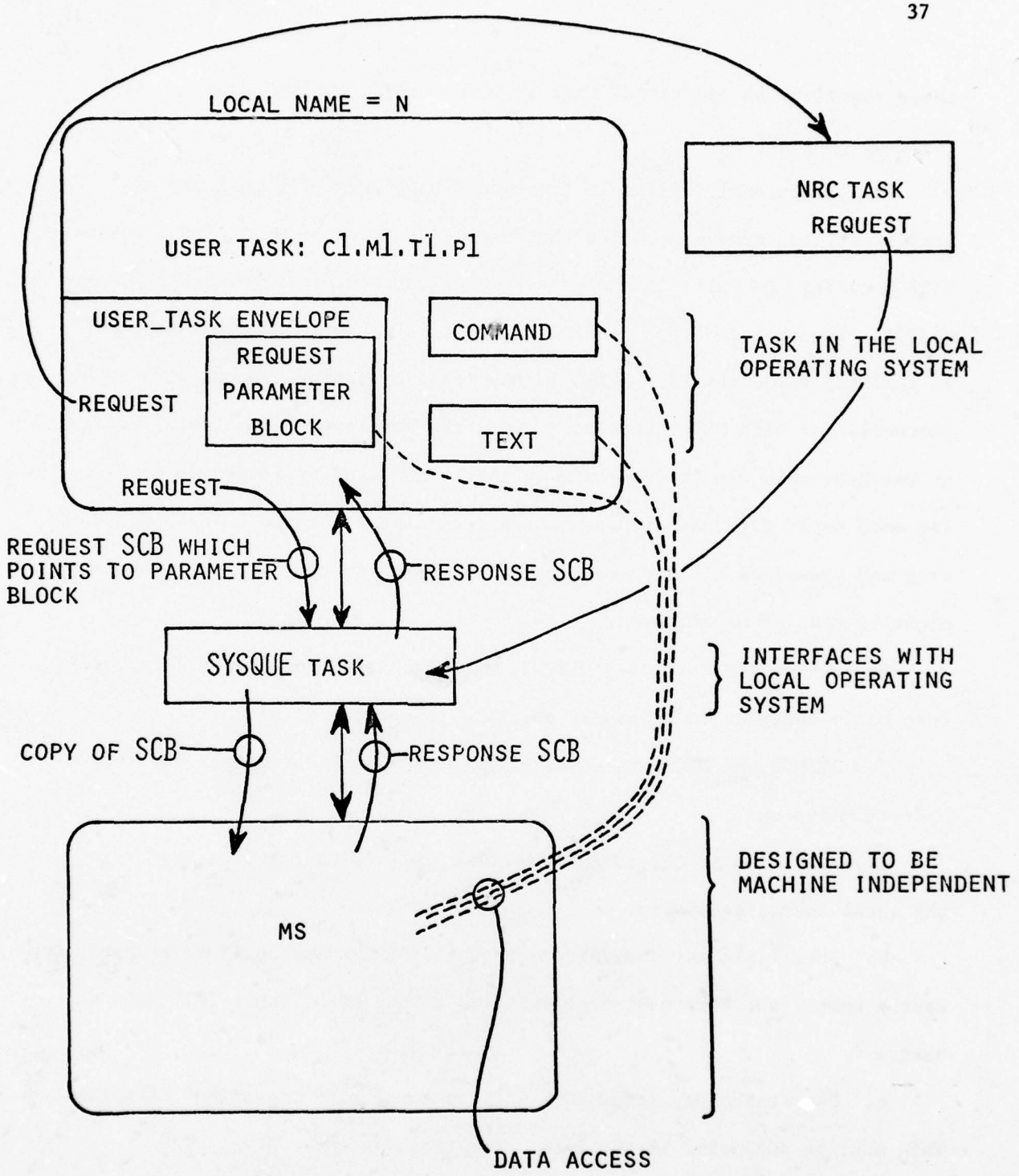


FIGURE 14: RELATION BETWEEN USER TASK AND MS

these functions an additional task is introduced: the NRC (for network resource controller).

For the initial design, it is assumed that each user task has a section of code appended called the "user task envelope." The user envelope will translate MS calls in the user task to appropriate intertask calls to SYSQUE. The user envelope requires a relatively knowledgeable author who is familiar with intertask calls in the local operating system, with SYSQUE protocol, and with MS conventions. Once the envelope is available, MS calls in the user task can be created with little concern for interface details. (At some later time, we propose that a special task should be created to trap and translate all user task calls to the MS so that the user envelope might be reduced or removed.)

Separating the envelope, SYSQUE, and NRC conventions as indicated, the core MS is designed to be easily portable to various machines.

The SYSQUE and NRC have several features which depend upon the local operating system:

a. A call to SYSQUE is an intertask call which requires support of the local operating system.

b. User tasks may communicate to the SYSQUE under their local operating system names, but they must be identified to the MS by their full network names.

c. User tasks may request to wait on completion of various MS operations. This must be supported by the local operating system.

d. If a user task terminates abnormally, the message system must be informed.

e. If a task, say c2.m2.t2, wants to start a task, say c1.m1.t1

(that is, in a different machine), that requires recourse to the local operating system in machine c1.m1.

5.2.3 Establishing a Network Name

Before a user task can use the message facility, it must establish a name by call to "NRC_ID." It is not permissible for the user task to assume a name without informing the NRC. When the user task is done with the message facility, it must relinquish its network name by a call to "NRC_RELEASE_ID."

5.2.4 Connection and Disconnection

Before a user task, with network name c1.m1.t1., can send and/or receive messages, it must mutually establish a logical connection with some other "twin" task. First, the user task selects a port name, say "p1," so that it can identify itself as "c1.m1.t1.p1" for communication with c2.m2.t2.p2. Hence, it must also know the name of the twin-task with which it will communicate. (The task c2.m2.t2.p2 is assumed to exist and also to request a connection to c1.m1.t1.p1. However, it is possible to request the NRC to initiate a twin task.)

Knowing these names, the task issues an "MS_CONNECT" request. It is an option to send a command packet (128 bytes) of data as part of the request; it is required to provide a buffer for receipt of 128 bytes of text from the other task. The mode types are discussed further later; both requests must specify corresponding modes. The two connection requests must occur within a time "window" (which is both a parameter of the message systems and a function of the physical separation between the MS's). Either task may issue the first request.

After issuing the connect request, the task may continue to execute other unrelated functions and it may periodically check either R or the

event variable, or it may wait upon the completion (or denial) of the request by issuing an "MS_WAIT."

Either of two connected tasks may break the connection by issuing an MS_DISCONNECT. Normally the disconnection request should not be issued until all send and receive requests have been completed. The GLOBAL mode breaks the connection entries in both MS's; the LOCAL mode only tags the connection entry in the local MS, which allows the twin task to finish receiving and processing the last message. The twin task must disconnect within a specified time window or else the tagged local disconnect will generate a GLOBAL disconnect.

5.2.5 Modes of Communication

Before specifying all of the send/receive requests, let us present several of the options in flow control for messages.

a. When a task passes a send/receive request to the MS, the request parameter block actually has a pointer to the message buffers (command and data). The request can either be queued to wait in the local MS or it can immediately initiate the protocol to start flow of the message. Thus, a queued receive buffer merely waits for an initiate send buffer to start the data flow (this is called rendezvous at the receiver); similarly, a queued send buffer merely waits for an initiate receive buffer to signal that the data flow may proceed (this is called rendezvous at the sender).

b. Because of varying loads in the local operating systems, MS requests are not processed instantaneously when they are issued. It is possible that an initiate request from one task can be transmitted to the twin task before the corresponding request is queued. To allow for this, the MS provides a "grace" period (on the order of say 20 seconds) for an initiate request to

wait for a matching request (which itself can be either queued or initiate); if the matching request is not logged within the grace period, the initiate request is denied (or aborted).

c. Messages within a cluster are always synchronized by the message system. That is, for an "initiate" send buffer, the command data buffer is first sent, and then an acknowledgment is returned to the sending MS to send the text buffer.

For messages between remote tasks, the user has the option that a send initiate can be sent without synchronization by the MS. That is, the buffers are packetized and sent without checking the receive buffers. The motivation for this is that there may be a long turn around time for packets over the remote line. Of course, then the receiving task is obligated to provide the n receiving buffers.

d. Rather than provide a queued message buffer to await an initiate request, it is also possible to inspect the request packet directly (by queuing an "ACCEPT_UNEXPECTED_COMMAND" request) and then to issue the appropriate queued send or receive request which will complete the message flow.

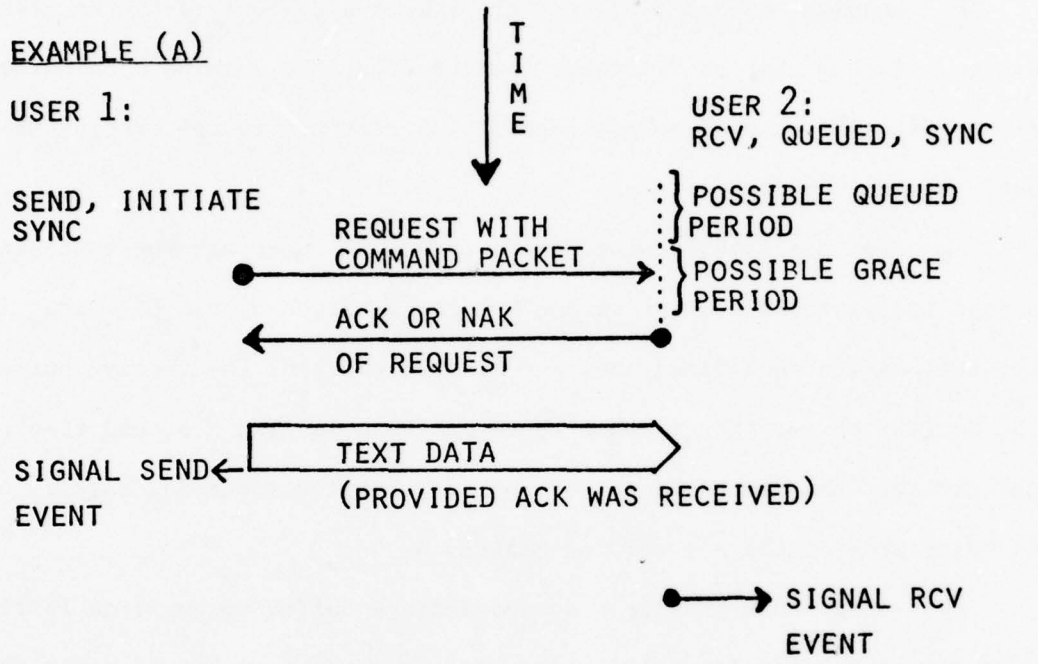
To illustrate these various options and their combinations, several message communication time sequence diagrams are shown in Figure 15.

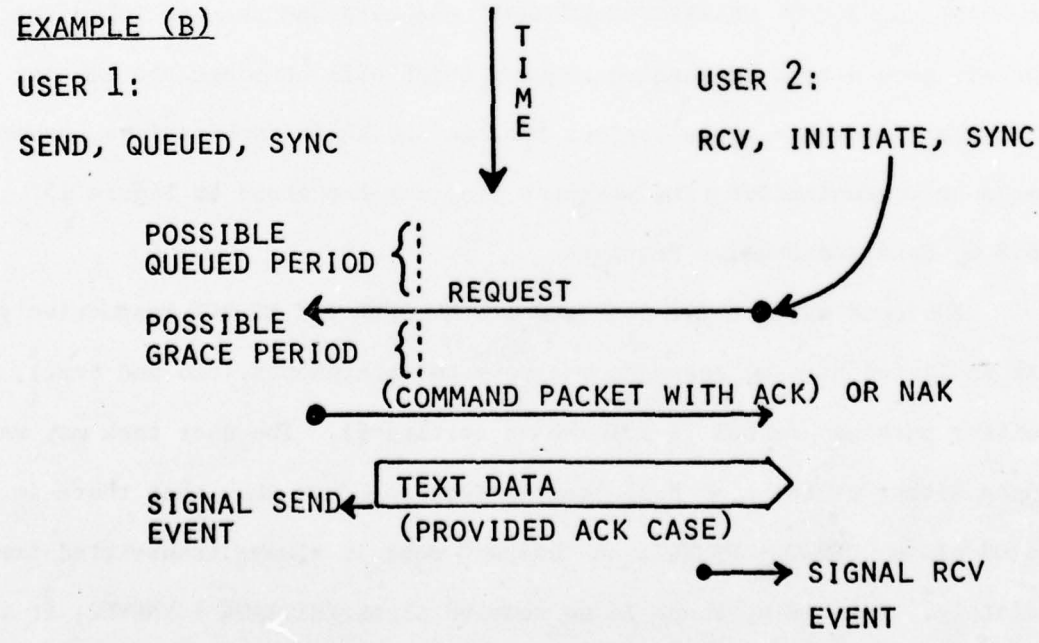
5.2.6 Send and Receive Requests

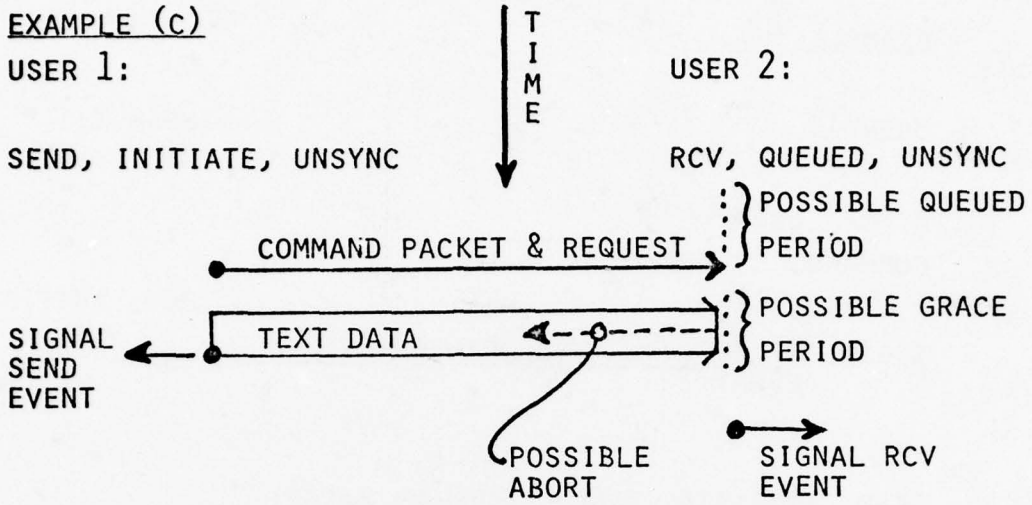
The send and receive requests are MS_SEND and MS_RCV respectively. As indicated before, messages may have two components (cmd and text), but either part may be null (cmdlen=0 or textlen=0). The user task may wait upon either of these as discussed in Section 5.2.4. Note that there is no send class QUEUED + UNSYNC; an "unsync" send is always transmitted immediately. Similarly, there is no receive class INITIATE + UNSYNC; it is presumed that the sender will initiate as soon as it is ready. If necessary in

FIGURE 15(1): MODES OF COMMUNICATION:

MESSAGE SEQUENCE DIAGRAMS






EXAMPLE (C)
USER 1:
TIME
USER 2:
SEND, INITIATE, UNSYNC
RCV, QUEUED, UNSYNC
POSSIBLE QUEUED
PERIOD
COMMAND PACKET & REQUEST
POSSIBLE GRACE
PERIOD
TEXT DATA
SIGNAL
SEND
EVENT
POSSIBLE
ABORT
SIGNAL RCV
EVENT

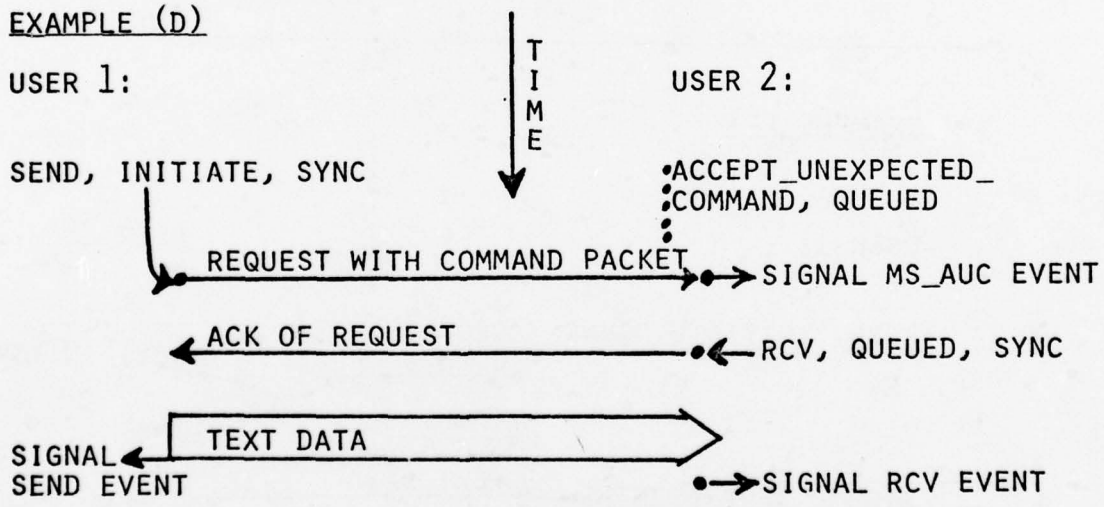
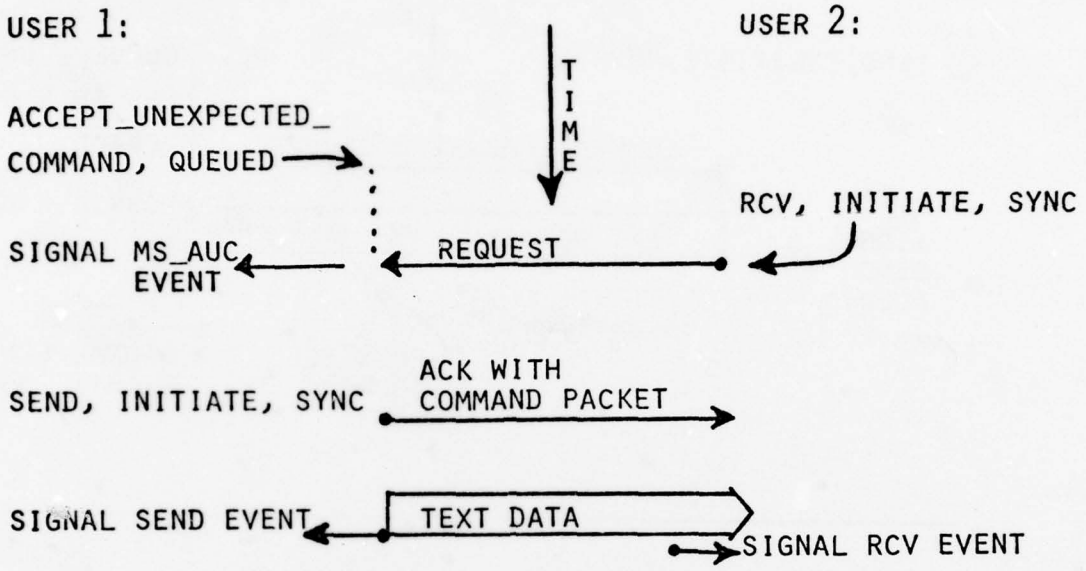

EXAMPLE (D)
USER 1:
TIME
USER 2:
SEND, INITIATE, SYNC
ACCEPT_UNEXPECTED_
COMMAND, QUEUED
REQUEST WITH COMMAND PACKET
SIGNAL MS_AUC EVENT
ACK OF REQUEST
RCV, QUEUED, SYNC
TEXT DATA
SIGNAL
SEND EVENT
SIGNAL RCV EVENT

FIGURE 15(2)

EXAMPLE (E)



EXAMPLE (F)

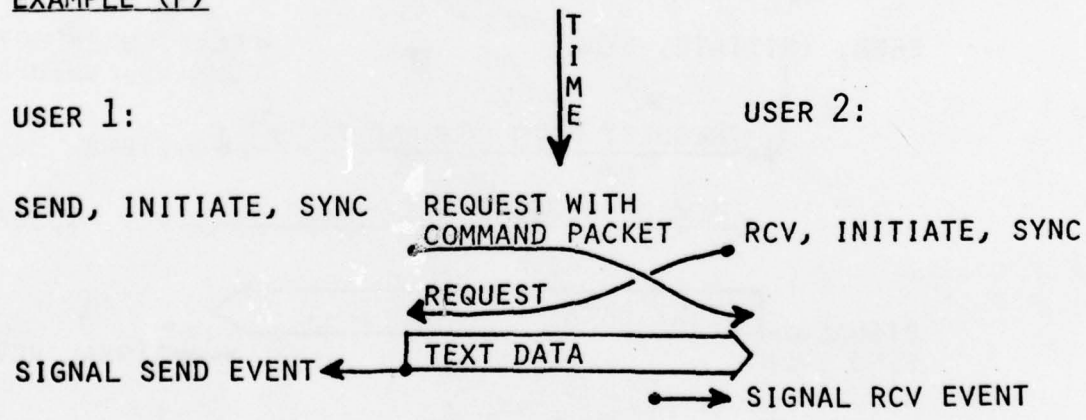


FIGURE 15(3)

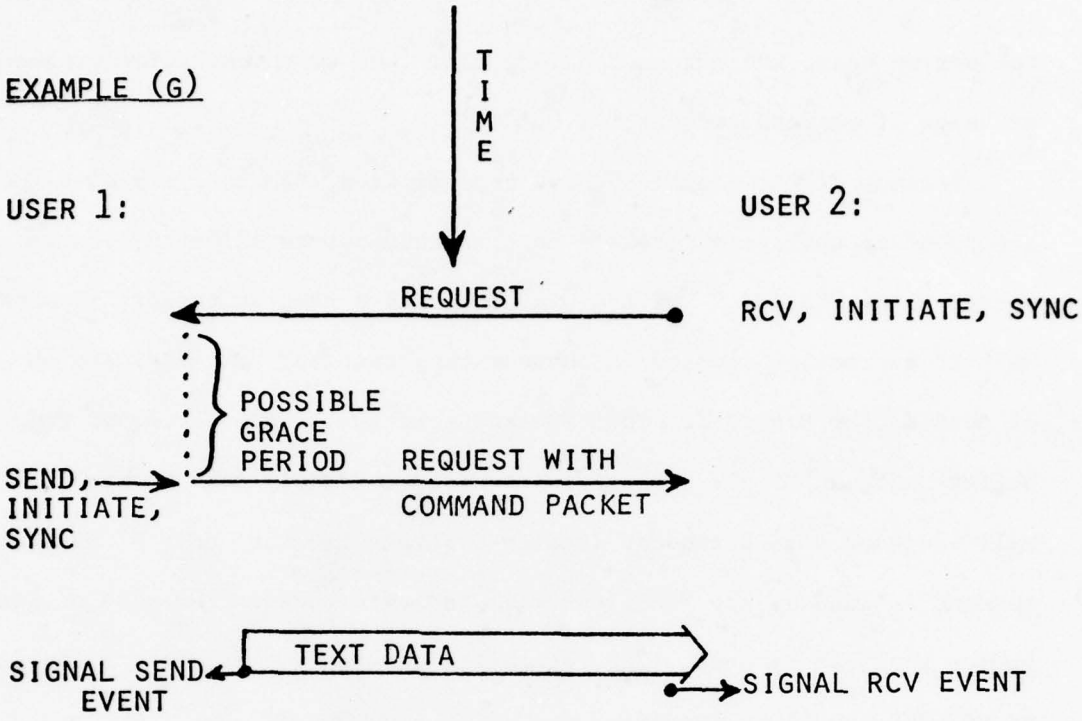

EXAMPLE (G)
TIME
USER 1:
USER 2:
REQUEST
RCV, INITIATE, SYNC
POSSIBLE
GRACE
PERIOD
REQUEST WITH
COMMAND PACKET
SEND,
INITIATE,
SYNC
TEXT DATA
SIGNAL SEND
EVENT
SIGNAL RCV EVENT

FIGURE 15(4)

the unsync case, user tasks can form their own synchronization protocol by exchange of command packets.

Because of the normal receive capabilities, the user may also "accept" a copy of an unexpected (really unsolicited, but anticipated) command packet using "MS_AUC." MS_AUC will indicate a receive request (type=RCV_REQ) only if no receive_command is queued; the "textlen" will indicate the size of text buffer required. The command packet will not be removed from the MS buffer pool; an MS_RCV request is required to receive the message. MS_AUC will indicate a send request (cmdlen=0, type=SEND_REQ) only if no send_ command is queued; the "textlen" will indicate the maximum size of text buffer that can be sent. The request is not removed from the MS tables; an MS_SEND should be issued.

5.2.7 Abnormal Cases

Several error cases have already been indicated in the response codes for the MS calls. If the user task recognizes that a bad message was requested to be sent/received, it can abort the message using MS_ABORT command.

Completion of a send command does not guarantee that the twin task successfully received the message (it could be lost if an intermediate machine crashes). Hence, user tasks should provide their own protocol for acknowledging successful receipt and processing of each message. Also, each sending task should backup each sent message at least until it is acknowledged by the receiving task.

If a message (or a packet) is inadvertently "lost" in the network, it may be possible to recover it. Each MS has a provision to "spool" misrouted packets. However, this feature may or may not be supported within the local operating system.

5.2.8 Query and Monitoring

Several facilities for interrogating the current status and past progress of each local message system are included as part of the design. Most of the queries are designed for executive purposes, not for user tasks.

5.2.9 Operator Functions

Various activities are automatically logged to an operator's console. The operator can request any of the query information. Under the local operating system, the operator should be able to load and start the various message system tasks: NRC, SYSQUE, MS tasks. Assuming these are active, the "message system ready for initialization" message should appear. Then the operator should be able to execute at the console any of the control functions.

5.3 MIMICS Hardware Architecture

5.3.1 Objectives

In support of the overall objectives of the project on "Functionally Distributed Computer Systems Development: Software and Systems Structure," the objectives of the MIMICS hardware are threefold:

1. To provide a means of transferring data and/or control information from one computer to another at a high rate of speed

2. To provide for maximum protection of the network against unauthorized access (either malicious or accidental) by users

3. To provide a reasonable level of flexibility and adaptability in the physical interconnections, including the ability to add or delete computers to or from the network at a later date.

5.3.2 Overall Architecture

Figure 16 illustrates the appearance of a "typical" MIMICS computer network to the average user, where the circled "nodes" represent central

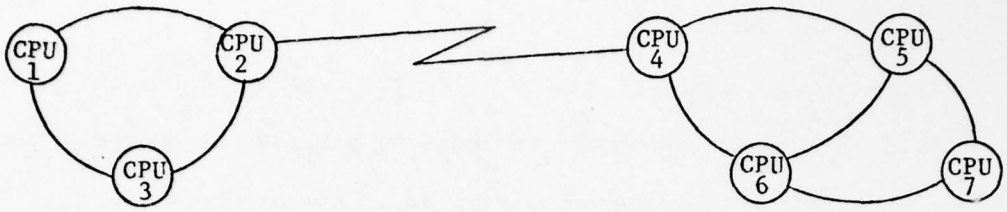


Figure 16

A "typical" computer network

processing units (CPU's) and the interconnecting lines represent possible information paths or links between the CPU's. In Figure 16, CPU's 1, 2, and 3 may be thought of as comprising a "local" network or cluster and CPU's 4, 5, 6, and 7 another cluster. Although not absolutely necessary, clusters "A" and "B" are probably "remote" from each other in some physical sense. Also note that each CPU in cluster "A" is physically connected to every other CPU in that same cluster (assuming that all inter-connection paths shown are operational at the moment), whereas cluster "B" is not so fully interconnected. This last observation implies that if some information must be transferred between CPU's 4 and 7, then this information must pass through CPU 5 or 6 or perhaps both; and, of course, if information transfer is desired between any CPU in cluster "A" and any CPU in cluster "B," then at least CPU's 2 and 4 must be involved.

For this project, one last point must be made: the rate of information transfer between CPU's in the same cluster will be very high (on the order of tens of millions of bits per second), while the rate of information transfer between clusters (such as between CPU's 2 and 4 in Figure 16) will be at a much lower value (on the order of five thousand bits per second). To reach this high intercluster transfer rate, CPU's in a cluster must be physically close together (on the order of tens of feet).

In actuality, the physical network interconnecting CPU's in the same

cluster is somewhat more complex than shown in Figure 16. In order to remove the burden of network control from the usually overworked main CPU and in order to provide a layer of hardware protection against either malicious or accidental interference between the network and the general users, the function of network control has been "unloaded" from the "host" CPU to another "control" CPU associated with each host. In a commercial network (but not in the prototype being developed by this project) these control CPU's would probably be "micro" computers (hence the acronym MIMICS for MIni MIcro Computer System) and their controlling software would probably be stored in Read-Only Memory (ROM), thus effectively precluding any unauthorized tampering by user programs executing in the host CPU's.

Figure 17 shows the interconnection for cluster "A" in greater detail.

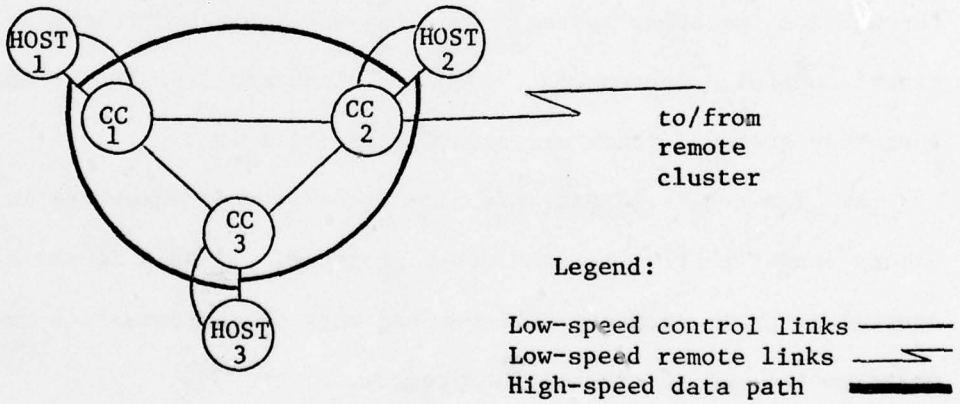


Figure 17

Local network "A" in greater detail than given in Figure 16

In this drawing, low-speed links are depicted by narrow lines and high-speed data paths by wide lines.

Note these points of interest:

a. Associated with each host CPU is a communication controller (CC).

b. Each host and CC pair are interconnected with a relatively slow-speed (9600 bits per second) control link as well as a very high-speed data path.

c. Each host/CC pair is connected to other host/CC pairs by similar slow-speed control and high-speed data links.

d. Connections to and from remote clusters pass through a CC and not a host CPU. For the sake of terminology, such connections are called "remote" links, and these links may carry both data and control information.

5.3.3 Data Flow

Although a detailed description of data and information flow will not be given in this review of the system, the general idea is as follows: Suppose a particular host CPU requests some data (perhaps in response to a request coming from a user application program executing on that host) through its operating system. This request is passed to the host's associated control computer via the control link and from thence to the network operating system. There are several possibilities:

a. The requested data may already be present somewhere in the host's memory (under control of some other program). If such is the case, the control computer arranges a high-speed copy of the data to a memory area under control of the requesting program.

b. Or if perchance the data is in the CC memory, the CC will arrange a similar high-speed memory-to-memory transfer of the data back to the requesting program.

c. If the data is not found in either the host or control computer memories, the CC will interrogate the cluster operating system concerning the requested data's whereabouts; if the data is found in some other CC

or host memory in the cluster, then the two CC's will arrange a high-speed memory-to-memory transfer of the data back to the requesting program. All of the above requests are transmitted over the local control links.

d. If the requested data is not found within the cluster, the request may be passed to other (remote) clusters via the remote link. If (when) the data is found in another cluster, then it is transferred back to the requesting host in at least two steps: first from the CC in the remote cluster to the CC in the requesting cluster (via the remote link) and then from that CC directly to the requesting host (via the high-speed path).

Several other cases could be examined, but the above examples should convey the general flavor of the operation of MIMICS. Neither the nature nor the form of the data and control information which is transmitted through the network is important at this level of understanding.

5.4 A Proposal for an Extensible-Contractible Network Control Language--ENCL

5.4.1 Extensible-Contractible Languages

An alternative to the current proliferation of special-purpose languages is a class of general-purpose languages. One approach to the creation of a general-purpose language is the extensible language method. It is characterized by a small base language which includes a minimal set of primitive functions. These primitives include definitional facilities which allow the language user to construct a special-purpose language suited to his own application.

An extensible language provides the definitional facilities to create new data objects, new operators, and new syntactic structures. Several of the more sophisticated extensible language systems (PPL and ECL) allow the user to create a new flow of control mechanism. Additionally, if the system includes a multi-tasking environment, the system scheduler is written in the

base language and may be accessed and rewritten by the user. This set of definitional facilities provides the language user with the tools to synthesize a wide variety of special-purpose languages.

The definitions of the syntax and semantics of extensions are accomplished by three techniques:

a. Paraphrase technique: Extensions are defined in terms of features currently known in the base language.

b. Orthophrase technique: Extensions are defined in terms of features not currently known in the base language.

c. Metaphrase technique: Extensions involve changing the interpretation of features currently in the language.

A problem can arise if a user is allowed to retain extensions for a prolonged period of time. The language and its associated translator can become large and inefficient. But if the translator is structured so that the features required for translating extensions may be inserted into and contracted out of the translator, the problem is solved. The result is an extensible-contractible language.

Designers of network control languages are faced with the problem of making the available hardware adaptable to a diverse user community. We will propose and specify an extensible-contractible network control language which provides a flexible, yet easy-to use interface. This language must be transportable to a heterogeneous set of machines with a minimum of effort.

5.4.2 ENCL

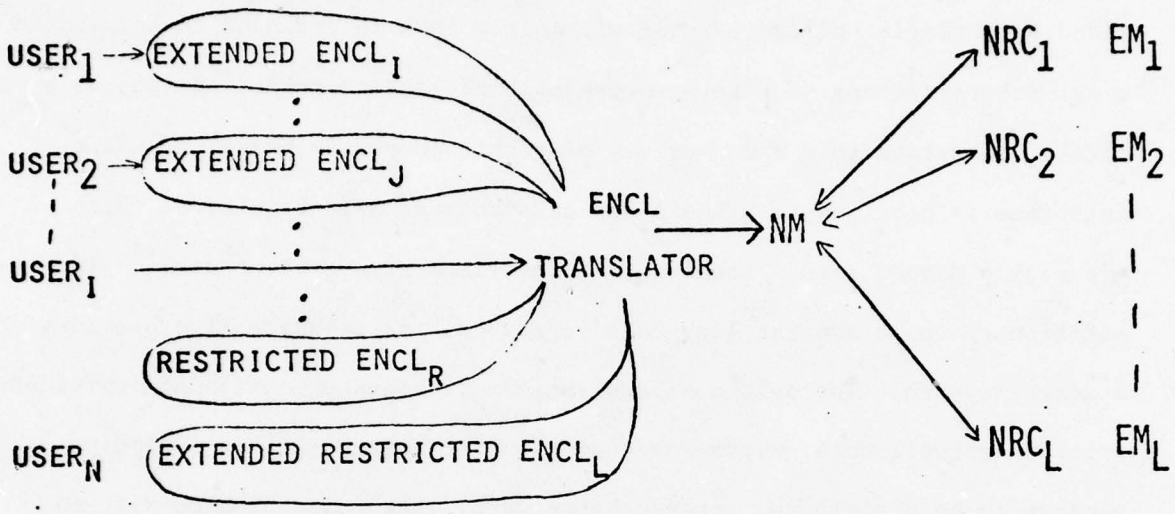
5.4.2.1 Design Objectives

The proposed language, ENCL, provides the network user with a programming tool which can be tailored to an individual's needs and level of competency. That is, ENCL can be made easy to use for the novice, or it can be made

powerful for the trustworthy professional. This flexible interface is provided by a single unified translator. Since ENCL is intended to be used on a network consisting of heterogeneous machine architectures, an adaptable form of interface to a wide variety of architectures is provided. This interface is provided by a small set of standard basic primitives which are easily mapped into a local operating system's supervisor calls. In addition to these adaptability considerations, the language also provides a secure system. The system allows the simultaneous execution of individual programs as well as simultaneous execution of jobsteps within a single program to be distributed across the network. The major problem with an implementation of a language which provides the flexibility of ENCL is that of translator efficiency. Although flexibility of the language is the primary objective, only the most efficient methods of providing this flexibility were included.

5.4.2.2 Design Approach

ENCL was designed so the semantic functions of the language can be distributed across the network. The ENCL system is structured to provide flexibility, distribution of functions, and portability. The flexible user interface is provided by a portable, extensible-contractible translator. The output from the translator is an intermediate language which is interpreted by a virtual network machine. This portable network machine (NM) provides the scheduling and synchronization necessary to allow simultaneous execution. The network machine produces standard network commands which drive the network's member machines. The member machines, called extended machines (EM), interface with the network machine through a small, machine dependent, network resource controller (NRC). Figure 18 illustrates a general view of the ENCL.



ENCL - EXTENCIBLE NETWORK CONTROL LANGUAGE
NM - NETWORK MACHINE
NRC - NETWORK RESOURCE CONTROLLER
EM - EXTENDED MACHINE

FIGURE 18: USER'S VIEW OF ENCL SYSTEM

5.4.3 ENCL Network Standard Commands

Network standard commands are classified into three categories: task control commands, file operation commands, and resource allocation commands. Task control commands provide the means of starting, stopping, pausing, and resuming tasks running under the control of an extended machine's local operating system. Task commands also provide the means of assigning files and/or a console to tasks. File operation commands provide the means of creating, deleting, and modifying files which are cataloged within an extended machine's local operating system's file directory. Resource allocation commands provide the means of allocating and deallocating an extended machine's resources which have been dedicated to the network. In general the operands for these commands are network standard task and file identifier names and network standard file descriptor blocks.

The following is a list of the network standard commands and their descriptions:

RUN Binds a network standard representation of a task identifier to the local representation of a task-id.

Binds a default or specified partition of memory with the task.

Creates an executable module of a specified routine and associates the module with a task control block.

Loads the module into the appropriate memory area.

Assigns a specified set of files and/or a console to the task to be executed.

Starts execution of the task.

STOP Terminates execution of the specified task.

PAUSE Causes execution of the specified task to be paused.

RESUME Causes execution of the specified task to be resumed.

CREATE Binds an external definition of a file descriptor block with a local description of a file descriptor block.

Allocates appropriate storage.

DELETE Deallocates storage and destroys the file descriptor block associated with the specified file-id.

MODIFY Modifies a specified file descriptor block.

LOOKUP Returns a specified file descriptor block.

PREALLOCATE Provides a means of preallocating disk and/or memory storage.

DEALLOCATE Releases allocated storage.

EXEC Provides machine dependent control language text.

Appendix A

Articles and Publications

Technical Reports, KSU

Number	Authors and Titles
CS 76-03	Maryanski and Wallentine. Implementation of a Distributed Data Base System. 18 pages. February, 1976.
CS 76-08	Fisher, Maryanski, and Wallentine. Evaluation of Conversion to a Back-End Data Management Systems. 18 pages. Published in the proceedings of ACM National Conference. October, 1976.
CS 76-11	Maryanski, Fisher, Wallentine, Calhoun, and Sernowitz. A Minicomputer Based Distributed Data Base System. 20 pages. April, 1976. Published in the proceedings of the NBS-IEEE Trends and Application Symposium: Micro and Mini Systems. May, 1976.
CS 76-12	Maryanski and Wallentine. A Simulation Model of a Back-End Data Base Management System. 24 pages. April, 1976. Published in the proceedings of the Seventh Annual Pittsburg Modeling and Simulation Conference. April, 1976.
CS 76-13	Maryanski. Language Specification for a Distributed Data Base Management System. 76 pages. May, 1976.
CS 76-14	Maryanski. Memory Management in a Distributed Data Base Management System. 48 pages. October, 1976.
CS 76-16	Neal, Anderson, Ratliff, and Wallentine. KSU Implementation of Concurrent PASCAL - A Reference Manual. 69 pages. December, 1976.
CS 76-17	Wallentine and McBride. Concurrent PASCAL - A Tutorial. 129 pages. December, 1976.

CS 76-18 — Hankley and Rawlenson. Sequential PASCAL Supplement for FORTRAN Programmers: A Primer of Slides. 145 pages. December, 1976.

CS 76-22 — Maryanski, Fisher and Wallentine. A User-Transparent Mechanism for the Distribution of a CODASYL Data Base Management System. 34 pages. December, 1976.

CS 77-1 — Maryanski. A Survey of Developments in Distributed Data Base Management Systems. February, 1977.

CS 77-2 — Maryanski. A Deadlock Prevention Algorithm for Distributed Data Base Management Systems. February, 1977.

CS 77-4 — Wallentine, Hankley, Anderson, Calhoun, and Maryanski. Progress Report on Functionally Distributed Computer Systems Development. 147 pages. December, 1976.

CS 77-5 — Maryanski and Fisher. Roll-back and Recovery in Distributed Data Base Systems. 19 pages. February, 1977.

CS 77-7 — Maryanski. Performance of Multi-processor Backend Data Base Systems. 15 pages. April, 1977.

CS 77-8 — Fisher and Maryanski. Design Considerations in Distributed Data Base Management Systems. 19 pages. April, 1977.

Appendix B

Reports

Date	Subject
March 24, 1976	Report of Monthly Review
April 21, 1976	Report of Monthly Review
May 20, 1977	Progress Report
September 1, 1976	Report of In-process Review
July 1, 1976	Progress Report to ARO
January 27, 1977	Progress Report to ARO

Appendix D

BIBLIOGRAPHY

[CAH 74] Hoare, C.A.R. Monitors: An Operating System Structuring Concept. Communications of ACM, Vol. 17, No. 10, (October) 1974.

[DNN 76a] Neal, D.N. An Architectural Base For Concurrent PASCAL. (M.S. Thesis) KSU Department of Computer Science. (in preparation) Tech. Rpt. CS76-19, Nov., 1976.

[DNN 76b] Neal, D.N., Anderson, G., Ratliff, J. and Wallentine, V. KSU Implementation of Concurrent PASCAL - A Reference Manual. CS76-16.

[GVB 75] Bochmann, G.V. Logical Verification and Implementation of Protocols. Fourth Data Communications Symposium, October, 1975.

[INT a] INTERDATA INC. 16 Bit Series Reference Manual. Pub. No. 29-398R03.

[INT b] INTERDATA INC. MODEL 8/32 Processor User's Manual. Pub. No. 29-428.

[INT c] INTERDATA INC. OS-32/MT Program Reference Manual. Pub. No. B29-390R02.

[JHH 76] Howard, J.H. Signaling in Monitors. Proceedings of 2nd Int. Conf. on Software Engineering (ACM/IEEE/NBS). (IEEE Cat. No. 76 CH1125-4 C) October, 1976.

[J&W 74] Jensen, K. and Wirth, N. PASCAL - User Manual and Report In Lecture Notes in Computer Science, No. 18 Spunger Verlog, 1974.

[NW 71] Wirth, N. The Programming Language PASCAL. ACTA Informatica Vol. 1, No. 1 (1971), pp. 35-63.

[PBH 73] Brinch Hansen, P. Concurrent Programming Concepts ACM Computing Surveys, Vol. 5, No. 4 (December), 1973.

[PBH 75a) Brinch Hansen, P. The Programming Language Concurrent PASCAL. IEEE Transactions on Software Engineering, Vol. 1, No. 2 (June, 1975), pp. 199-207.

[PBH 75b] Brinch Hansen, P. Concurrent PASCAL Report. Information Science, California Institute of Technology, June, 1975.

[PBH 76] Brinch Hansen, P. The SOLO Operating System. Software-Practice and Experience, Vol. 6, No. 2 (April-June, 1976) pp. 141-206.

[WJH 76] Hankley, W.J. et. al. Sequential PASCAL Supplement (for FORTRAN Programmers). KSU Department of Computer Science. Tech. Rpt. CS76-18, Nov., 1976.

(2) remote - two machines (or user tasks) are remote if communication between them must travel over low speed telecommunications lines (eg. 2400 baud, synchronous); messages between remote tasks are packetized by the message system, ie., broken into packets for transmission and reconstructed at the receiving machine; opposite of local.

(2) local - two user tasks are local if either (i) they are in the same machine or (ii) they are in machines connected by high speed "data movers" (eg. 2 million bits per sec); messages between local tasks are not packetized, then are sent as a block, memory-to-memory using the data movers; each group of local machines is called a cluster; opposite of remote.

(1) host - any computer in the network with user tasks in it; warning this differs from usual data-base terminology as in a distributed data base application, both the front-end and back-end computers would be called network hosts; in MIMICS hosts may be either minicomputers or maxicomputers.

(2) off-loading - the removing of some operating system or language support functions from a host machine to an allied dedicated processor; the motivation for this is that the off-loaded functions can execute truly concurrently (ie., simultaneously) with tasks in the host, thus greatly improving the performance of the host; in MIMICS, the message system is typically off-loaded into a communications-control (CC) microprocessor; in the 370 architecture, the I/O functions are off-loaded to special channel-control processors.

(2) CC - communciation controller; a micro processor used in MIMICS for off-loading the message system from a host machine.

(1) message - basic unit of network communication; copied by the message system from address space of a sender user-task into agreed upon place in address space of a receiver user-task; in MIMICS, messages may have two components, (i) a command part (up to 128 bytes of data) and (ii) a data part (up to 64K bytes), but either (not both) of the parts may be null.

(2) routing - selection of the path between two host machines over which communication will flow - hence, the selection of (i) which intermediate machines, if any, are part of the path and (ii) which actual communication line, in case there is more than one, to use between any two directly connected machines; in MIMICS each message system instance has a route

table with entries ⟨name_of_another_machine_in_the_network: line_route_to_next_machine_in_the_path⟩ where the line-route number is a logic-line, so that all physical lines to an adjacent machine are used interchangeably.

(3) logical line - a group of parallel physical communications lines which directly connect two adjacent computers, where the actual physical lines are used interchangeably; warning this means that packets can flow "out-of-sequence," although user tasks never observe this phenomenon.

(3) KSUBUS - a special multiplexed hardware bus, designed by M. Calhoun at KSU, to form a memory-speed connection between a CC, one or two hosts which are on the bus, an XR -data-mover, and X- and R- data mover pairs which connect to other KSUBUS's in the same cluster.

(1) cluster - in MIMICS, a group of network machines that are all interconnected by high speed data movers; the data-parts of messages move at memory speeds from the sender task to the receiver task.

(3) c-node - a cluster-node; the group of one or two hosts which are connected to the same KSUBUS; messages can move memory-to-memory within a c-node without accompanying cluster protocol; warning in conventional network terminology, any machine in the network would be called a node, but that is different from the c-node concept.

(3) data-mover - a special "autonomous functional hardware unit," designed at KSU, to work in conjunction with other matching units to move data blocks memory-to-memory at memory speeds between machines in the same cluster; XR-, X- and R- data movers; a data mover can be enabled only by the cc on the same KSUBUS as the data mover.

(3) XR - data mover - device which copies a block of data from one area to another within machines on the same KSUBUS, eg. host-to-host or host-to-cc or cc-to-host.

(3) X-data mover - device which "transmits" a block of data to an R-unit on a connected KSUBUS, where the source of the data is either (i) memory of a machine on the same KSUBUS with the X-unit (called X-mitting) or (ii) an R-unit on the same KSUBUS as the X-unit (which is called forwarding of the data).

(3) R-data mover - device which receives data from an X-unit on a connected KSUBUS and "moves" the data to either (i) memory of a

machine on the same KSUBUS as the R-unit (called receiving) or (ii) to an X-unit on the same KSUBUS (called forwarding).

(3) packet - a basic unit for communication over a low speed line; in MIMICS the packets have the following components:

beginning_part = 6-SYN's - DLE - STX

packet_flow_control (4 bytes) = RC---return control character
RN---return sequence character
N----out sequence character
TL---text_length character

message_flow_control (12 bytes) = SEQ---packet sequence number (2 bytes)
T-----type of packet character
ID----message id character
TO_ID---4 bytes
FROM_ID---4 bytes

packet_text (0 to 128 bytes of data character plus transparency characters as required plus extra SYN characters as needed)

check_sum_part (2 bytes)

end_part = DLE-ETX

This comprises normally up to 156 characters, and most likely several more, to transmit data text of up to 128 bytes, so that the effective line baud rate is less than the nominal baud rate. Transmission errors and subsequent retransmission reduce the effective line baud rate even further.

(2) buffering - mechanism for providing space (buffers, actually "empty" buffers) and temporarily storing information (also called buffers, or full buffers), so that the related steps of storing and removing buffers (actual contents of the buffers) can proceed asynchronously, with the cumulative number of stores at all times ahead of the cumulative number of removals.

Buffers in MIMICS include

(2) SYSQUE --- buffer between user tasks and message system; buffers requests to message system and responses back to user tasks.

protocol--an agreed upon form and sequence for exchange of control information and data between processes to achieve a synchronized communication, i.e. so that the information is correctly conveyed and both processes know it; there are several sets of protocol in MIMICS, including:

(1) SYSQUE_protocol - protocol for both user tasks and message system to both send and receive SCB's, which are control blocks used to implement passing of parameter information for message requests and responses.

(1) Message_system - set of paramters lists for message system requests together with rules for acceptable user-task behavior.

(3) Synchronous_line - rules of sequencing for exchanging packets between remote line drivers.

(3) CC_protocol - actually two sets of protocols;

(i) rules for exchanging packets between cluster - CC's (same as synchronous line protocol) and

(ii) rules for controlling the data mover's copying of data blocks within the cluster.

(3) PASCAL - a programming language designed by N. Wirth which promotes correct programs because (i) it promotes structured programs (both flow of control and data structures) and (ii) it enforces numerous compile time checks not normally supported in other programming languages (thus minimizing run-time errors), and (iii) it allows code to be written in a very easily readable form.

(3) PASCAL - at the same time, a restriction of PASCAL to enforce simple programs and an extension of PASCAL to support a well structured mechanism for concurrent programs using monitors; developed by P. Brich-Hansen; ported to KSU for use in implementing a readable and correct prototype of the message system.

(3) monitor - a concept introduced by C. Hoare for structured programming of concurrent processes; the monitor consists of (i) a group of shared data structures, (ii) a set of procedures (monitor entry points) which operate on the shared data, (iii) and initial state for the shared data, and (iv) the convention that only one process may execute "in" the monitor at any one time, so that the programmer does not have to worry about difficulties of multiple processes writing to the shared data at the same time; monitors are implemented in CPASCAL; monitors in the MIMICS implementation include:

- SYSQUE - monitor of SCB's for message system requests and responses.
- Packet_buffer - monitor of packets to be sent or just received.
- MESS_table - monitor of active and queued SEND and RCV requests.
- CONNECT_table - monitor of user task connection status information.
- logical_line_window - monitor of packets actively being transmitted, received, or acknowledged over low speed lines; one for each logical line.
- event_control - monitors to control a process which has to await availability of data in either of two (or more) other monitors, since a process in CPASCAL can normally wait on only one monitor.
- cluster_monitor - monitor of request and responses for activation of the data movers.

(1) NRC - Network Resource Controller; a network operating system module needed to interface user tasks between the local operating system in the host machine and the network operating system; one for each host machine; functions of this NRC include:

- supplying network names to each user task.
- initiating tasks in a host upon request from the NRC in some other host (based up requests from user programs)
- disconnecting user tasks from the message system when the task terminates without the normally expected disconnect step.

(1) local_operating_system - the regular operating system in any single host machine.

(1) network operating system - the collection of all operating software in all network machines including all NRC's, all message system instances, all SYSQUE's, etc.

(2) user envelope - interface software to translate message system calls in user programs to appropriate usage of the SYSQUE; in particular, the user envelope will need to supply specific network names for all communications requests.

(1) network-names (c.m.t.p.) - all communications in MIMICS is directed using a network wide naming convention consisting of four bytes:

c = cluster character

m = machine

t = unique task identifying character, within machine c.m.

p = port character: the port character effectively identifies a communication subname so that one task may carry one network communication using two different ports and keep messages to each port separate.

c, m, and t names for a task can be established by interrogating the NRC.

(1) local names - within a host tasks will be identified by names assigned by the local operating system; these are not network names; warning it is necessary to translate between local names and network names in order to interface user tasks to both the local operating system and the network operating system.

(2) Back-end - typically refers to a host computer executing only a data base management function. Sometimes refers to the function inside a partition in a host which executes application programs in other partitions.

(3) packet_buffer_monitor --- buffers packets to be sent over low speed lines and received from a low speed line.

(3) line_drivers --- buffers the packets as they are actually being transmitted or received over a low speed line.

(1) message_table --- buffers SEND and RCV requests that have been accepted by the message system but not yet completed.

Appendix C

TABLE 1 Vocabulary

In discussing the MIMICS network concepts and implementation it is essential to establish certain base vocabulary. Several of these key words are explained in the list which follows. Each word has been graded using the scheme.

(1) ------ means word is essential for network users

(2) ------ means word is needed for discussion of network concepts

(3) ------ word is related to network implementation.

(1) network - an interconnected set of computers

(1) MIMICS - a network designed to be implemented using mini- and micro computers, but also with larger machines in the network; developed at KSU under support from the U.S. Army, Computer Systems Command.

(1) connected - the network hardware is said to be connected if it is possible for communication to flow from any one machine to any other machine in the network, either directly or indirectly via intermediate machines; MIMICS is intended to be connected.

- two user tasks are said to be connected if they have mutually established a "logical connection" by appropriate matching MS_CONNECT calls; these tasks may then communicate using MS_SEND and MS_RCV calls.

(1) user task - an application task in one of the network host machines that communicates to some other user task, likely in a different machine, using the message system.

(1) message system - that software/hardware part of MIMICS that supports network communication by user tasks; basic message system commands are CONNECT, DISCONNECT, SEND, and RCV (receive); basic message system functions are routing of messages, packetizing messages for remote transmissions, buffering of packets, handling of line protocol for packets and messages, and reconstruction of packetized messages.

[MB 76] Ball, Mike. Personal Communication. Naval Underseas Laborary, San Diego, CA.

[FJM 76] Maryanski, F.J. Design Considerations for a Distributed Data Base Management System, TR CS-76-14, Computer Science Department. Kansas State University. September 1976.